

Sintaxis y procesamiento de cifrado XML

Recomendación del W3C del 10 de diciembre de 2002

Esta versión:
http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/
Ultima versión:
http://www.w3.org/TR/xmlenc-core/
Versión previa:
http://www.w3.org/TR/2002/PR-xmlenc-core-20021003/
Editores
Donald Eastlake <dee3@torque.pothole.com>
José Reagle <reagle@w3.org>
Autores
Takeshi Imamura <IMAMU@jp.ibm.com>
Blair Dillaway <blaird@microsoft.com>
Ed Simón <edsimon@xmlsec.com>
Colaboradores
Ver participantes .

Consulte las **erratas** de este documento, que pueden incluir algunas correcciones normativas. Ver también **traducciones** .

Abstracto

▾ colapsar

¡Esta versión es antigua!
Para obtener la última versión, consulte https://www.w3.org/TR/xmlenc-core1/.

Este documento espec... ser datos arbitrarios (incluido un documento XML), u... un elemento de cifrado XML que contiene o hace re...

Estado de este documento

Este documento es la Recomendación de cifrado XML (REC) del W3C . Este documento ha sido revisado por miembros del W3C y otras partes interesadas y ha sido respaldado por el Director como Recomendación del W3C. Es un documento estable y puede usarse como material de referencia o citarse como referencia normativa de otro documento. El papel del W3C al elaborar la Recomendación es llamar la atención sobre la especificación y promover su implementación generalizada. Esto mejora la funcionalidad y la interoperabilidad de la Web.

Esta especificación fue producida por el Grupo de Trabajo de Cifrado XML del W3C (Actividad), que cree que la especificación es suficiente para la creación de implementaciones interoperables independientes como se demuestra en el Informe de Interoperabilidad.

Las divulgaciones de patentes relevantes para esta especificación se pueden encontrar en la página de divulgación de patentes del Grupo de Trabajo de conformidad con la política del W3C.

Informe los errores en este documento a xml-encryption@w3.org (archivo público).

La lista de errores conocidos en esta especificación está disponible en http://www.w3.org/Encryption/2002/12-xmlenc-errata .

La versión en inglés de esta especificación es la única versión normativa. La información sobre las traducciones de este documento (si corresponde) está disponible en http://www.w3.org/Encryption/2002/12-xmlenc-translations .

A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR/.

Table of Contents

---

1 Introduction

This document specifies a process for encrypting data and representing the result in XML. The data may be arbitrary data (including an XML document), an XML element, or XML element content. The result of encrypting data is an XML Encryption `EncryptedData` element which contains (via one of its children's content) or identifies (via a URI reference) the cipher data.

When encrypting an XML element or element content the `EncryptedData` element replaces the element or content (respectively) in the encrypted version of the XML document.

When encrypting arbitrary data (including entire XML documents), the `EncryptedData` element may become the root of a new XML document or become a child element in an application-chosen XML document.

1.1 Editorial and Conformance Conventions

This specification uses XML schemas [XML-schema] to describe the content model.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be interpreted as described in RFC2119 [KEYWORDS]:

> "they MUST only be used where it is actually required for interoperation or to limit behavior which has potential for causing harm (e.g., limiting retransmissions)"

Consequently, we use these capitalized keywords to unambiguously specify requirements over protocol and application features and behavior that affect the interoperability and security of implementations. These key words are not used (capitalized) to describe XML grammar; schema definitions unambiguously describe such requirements and we wish to reserve the prominence of these terms for the natural language descriptions of protocols and features. For instance, an XML attribute might be described as being "optional." Compliance with the XML-namespace specification [XML-NS] is described as "REQUIRED."

1.2 Design Philosophy

The design philosophy and requirements of this specification (including the limitations related to instance validity) are addressed in the XML Encryption Requirements [EncReq].

1.3 Versions, Namespaces, URIs, and Identifiers

No provision is made for an explicit version number in this syntax. If a future version is needed, it will use a different namespace. The experimental XML namespace [XML-NS] URI that MUST be used by implementations of this (dated) specification is:

```
xmlns:xenc='http://www.w3.org/2001/04/xmlenc#'
```

This namespace is also used as the prefix for algorithm identifiers used by this specification. While applications MUST support XML and XML namespaces, the use of internal entities [XML, section 4.2.1], the "`xenc`" XML namespace prefix [XML-NS, section 2] and defaulting/scoping conventions are OPTIONAL; we use these facilities to provide compact and readable examples. Additionally, the entity `&xenc;` is defined so as to provide short-hand identifiers for URIs defined in this specification. For example "`&xenc;Element`" corresponds to "http://www.w3.org/2001/04/xmlenc#Element".

This specification makes use of the XML Signature [XML-DSIG] namespace and schema definitions

```
xmlns:ds='http://www.w3.org/2000/09/xmldsig#'
```

Los URI [URI] DEBEN cumplir con la definición de tipo [XML-Schema] y la especificación [XML-DSIG , 4.3.3.1 El atributo URI] (es decir, caracteres permitidos, escape de caracteres, soporte de esquema, etc.).`anyURI`

1.4 Agradecimientos

Se agradecen las contribuciones de los siguientes miembros del Grupo de Trabajo a esta especificación de acuerdo con las políticas de contribuyentes y la lista activa del Grupo de Trabajo .

- joseph ashwood
- Simon Blake-Wilson, Certicom
- Frank D. Cavallito, Sistemas BEA
- Eric Cohen, PricewaterhouseCoopers
- Blair Dillaway, Microsoft (Autor)
- Blake Dournaee, Seguridad RSA
- Donald Eastlake, Motorola (Editor)
- Barb Fox, Microsoft
- Christian Geuer-Pollmann, Universidad de Siegen
- Tom Gindin, IBM
- Jiandong Guo, Faos
- Phillip Hallam-Baker, Verisign
- Amir Herzberg, NewGenPay
- Merlín Hughes, Baltimore
- Federico Hirsch
- Maryann Hondo, IBM
- Takeshi Imamura, IBM (Autor)
- Mike Just, Entrust, Inc.
- Brian La Macchia, Microsoft
- Hiroshi Maruyama, IBM
- John Messing, Ley en línea
- Shivaram Mysore, Sun Microsystems
- Thane Plambeck, Verisign
- Joseph Reagle, W3C (Presidente, Editor)
- Alexei Sanin
- Jim Schaad, consultoría Soaring Hawk
- Ed Simon, XMLsec (Autor)
- Daniel Toth, Ford
- Yongge Wang, Certicom
- Steve Wiley, mi prueba

Además, agradecemos a las siguientes personas por sus comentarios durante y después de la última llamada:

- Martín Durst, W3C
- Dan Lanz, Zolera
- Susan Lesch, W3C
- David Orchard, Sistemas BEA
- Ronald Rivest, MIT

2 Descripción general y ejemplos de cifrado (no normativo)

Esta sección proporciona una descripción general y ejemplos de la sintaxis de cifrado XML. La sintaxis formal se encuentra en *Sintaxis de cifrado* (sección 3); el procesamiento específico se proporciona en *las Reglas de procesamiento* (sección 4).

Expresado en forma abreviada, el `EncryptedData`elemento tiene la siguiente estructura (donde "?" denota cero o una ocurrencia; "+" denota una o más ocurrencias; "*" denota cero o más ocurrencias; y la etiqueta de elemento vacía significa que el elemento debe ser vacío):

```
<¿Identificación de datos cifrados? ¿Tipo? ¿Tipo de Mimica? ¿Codificación?>
  <Método de cifrado/>?
  <ds:Información clave>
    <Clave cifrada>?
    <Método de acuerdo>?
    <ds:NombreClave>?
    <ds:Método de recuperación>?
    <ds:*>?
  </ds:KeyInfo>?
  <Datos cifrados>
    <ValorCifrado>?
    <¿URI de referencia de cifrado?>?
  </CipherData>
  <Propiedades de cifrado>?
</EncryptedData>
```

El `CipherData`elemento envuelve o hace referencia a los datos cifrados sin procesar. Si es envolvente, los datos cifrados sin procesar son el `CipherValue`contenido del elemento; si se hace referencia, el atributo `CipherReference`del elemento `URI`apunta a la ubicación de los datos cifrados sin procesar

2.1 Granularidad del cifrado

Considere la siguiente información de pago ficticia, que incluye información de identificación e información apropiada para un método de pago (por ejemplo, tarjeta de crédito, transferencia de dinero o cheque electrónico):

```
<?xml versión='1.0'?>
<PaymentInfo xmlns='http://ejemplo.org/pagov2'>
  <Nombre>John Smith</Nombre>
  <Límite de tarjeta de crédito='5000' Moneda='USD'>
    <Número>4019 2445 0277 5567</Número>
    <Emisor>Banco de ejemplo</Emisor>
    <Vencimiento>04/02</Vencimiento>
  </Tarjeta de crédito>
</PagoInfo>
```

Este margen representa que John Smith está usando su tarjeta de crédito con un límite de $5,000USD.

2.1.1 Cifrar un elemento XML

¡El número de tarjeta de crédito de Smith es información confidencial! Si la aplicación desea mantener esa información confidencial, puede cifrar el `CreditCard`elemento:

```
<?xml versión='1.0'?>
<PaymentInfo xmlns='http://ejemplo.org/pagov2'>
  <Nombre>John Smith</Nombre>
  <Tipo de datos cifrados='http://www.w3.org/2001/04/xmlenc#Element'
   xmlns='http://www.w3.org/2001/04/xmlenc#'>
    <Datos cifrados>
      <CipherValue>A23B45C56</CipherValue>
    </CipherData>
  </EncryptedData>
</PagoInfo>
```

Al cifrar todo el `CreditCard`elemento desde sus etiquetas de inicio a fin, se oculta la identidad del elemento en sí. (Un espía no sabe si utilizó una tarjeta de crédito o una transferencia de dinero). El `CipherData`elemento contiene la serialización cifrada del `CreditCard`elemento.

2.1.2 Cifrado del contenido del elemento XML (Elementos)

Como escenario alternativo, puede ser útil para los agentes intermediarios saber que John usó una tarjeta de crédito con un límite particular, pero no el número, el emisor y la fecha de vencimiento de la tarjeta. En este caso, el contenido (datos de caracteres o elementos secundarios) del `CreditCard`elemento está cifrado:

```
<?xml versión='1.0'?>
<PaymentInfo xmlns='http://ejemplo.org/pagov2'>
  <Nombre>John Smith</Nombre>
  <Límite de tarjeta de crédito='5000' Moneda='USD'>
    <EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
     Tipo ='http://www.w3.org/2001/04/xmlenc#Content'>
      <Datos cifrados>
        <CipherValue>A23B45C56</CipherValue>
      </CipherData>
    </EncryptedData>
  </Tarjeta de crédito>
</PagoInfo>
```

2.1.3 Cifrado del contenido del elemento XML (datos de caracteres)

O considere el escenario en el que toda la información, *excepto* el número de tarjeta de crédito real, puede estar clara, incluido el hecho de que existe el elemento Número:

```
<?xml versión='1.0'?>
<PaymentInfo xmlns='http://ejemplo.org/pagov2'>
  <Nombre>John Smith</Nombre>
  <Límite de tarjeta de crédito='5000' Moneda='USD'>
    <Número>
      <EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
       Tipo ='http://www.w3.org/2001/04/xmlenc#Content'>
        <Datos cifrados>
          <CipherValue>A23B45C56</CipherValue>
        </a> CipherDat_
      </a> EncryptedDat_
    </Número>
    <Emisor>Banco de ejemplo</Emisor>
    <Vencimiento>04/02</Vencimiento>
  </Tarjeta de crédito>
</PagoInfo>
```

Ambos `CreditCard`y `Number`están claros, pero el contenido de los datos de caracteres `Number`está cifrado.

2.1.4 Encrypting Arbitrary Data and XML Documents

If the application scenario requires all of the information to be encrypted, the whole document is encrypted as an octet sequence. This applies to arbitrary data including XML documents.

```
<?xml version='1.0'?>
<EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
 MimeType='text/xml'>
  <CipherData>
    <CipherValue>A23B45C56</CipherValue>
```

```
  </CipherData>
</EncryptedData>
```

2.1.5 Super-Encryption: Encrypting EncryptedData

An XML document may contain zero or more `EncryptedData` elements. `EncryptedData` cannot be the parent or child of another `EncryptedData` element. However, the actual data encrypted can be anything, including `EncryptedData` and `EncryptedKey` elements (i.e., super-encryption). During super-encryption of an `EncryptedData` or `EncryptedKey` element, one must encrypt the entire element. Encrypting only the content of these elements, or encrypting selected child elements is an invalid instance under the provided schema.
For example, consider the following:

```
<pay:PaymentInfo xmlns:pay='http://example.org/paymentv2'>
  <EncryptedData Id='ED1' xmlns='http://www.w3.org/2001/04/xmlenc#'
   Type='http://www.w3.org/2001/04/xmlenc#Element'>
    <CipherData>
      <CipherValue>originalEncryptedData</CipherValue>
    </CipherData>
  </EncryptedData>
</pay:PaymentInfo>
```

A valid super-encryption of "`//xenc:EncryptedData[@Id='ED1']`" would be:

```
<pay:PaymentInfo xmlns:pay='http://example.org/paymentv2'>
  <EncryptedData Id='ED2' xmlns='http://www.w3.org/2001/04/xmlenc#'
   Type='http://www.w3.org/2001/04/xmlenc#Element'>
    <CipherData>
      <CipherValue>newEncryptedData</CipherValue>
    </CipherData>
  </EncryptedData>
</pay:PaymentInfo>
```

where the `CipherValue` content of '`newEncryptedData`' is the base64 encoding of the encrypted octet sequence resulting from encrypting the `EncryptedData` element with `Id='ED1'`.

2.2 `EncryptedData` and `EncryptedKey` Usage

2.2.1 `EncryptedData` with Symmetric Key (`KeyName`)

```
[s1] <EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
      Type='http://www.w3.org/2001/04/xmlenc#Element'/>
[s2]   <EncryptionMethod
        Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
[s3]   <ds:KeyInfo xmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
[s4]     <ds:KeyName>John Smith</ds:KeyName>
[s5]   </ds:KeyInfo>
[s6]   <CipherData><CipherValue>DEADBEEF</CipherValue></CipherData>
[s7] </EncryptedData>
```

`[s1]`El tipo de datos cifrados se puede representar como un valor de atributo para ayudar en el descifrado y el procesamiento posterior. En este caso, los datos cifrados eran un "elemento". Otras alternativas incluyen el "contenido" de un elemento o una secuencia de octetos externa que también puede identificarse mediante los atributos `MimeType`y `Encoding`.

`[s2]`Este (3DES CBC) es un cifrado de clave simétrica.

`[s4]`La clave simétrica tiene un nombre asociado "John Smith".

`[s6] CipherData`contiene un `CipherValue`, que es una secuencia de octetos codificada en base64. Alternativamente, podría contener un `CipherReference`, que es una referencia de URI junto con las transformaciones necesarias para obtener los datos cifrados como una secuencia de octetos.

2.2.2 `EncryptedKey` (`ReferenceList`, `ds:RetrievalMethod`, `CarriedKeyName`)

La siguiente `EncryptedData`estructura es muy similar a la anterior, excepto que esta vez se hace referencia a la clave mediante `ds:RetrievalMethod`:

```
[t01] <Id. de datos cifrados='ED'
       xmlns='http://www.w3.org/2001/04/xmlenc#'>
[t02] <Método de cifrado
         Algoritmo='http://www.w3.org/2001/04/xmlenc#aes128-cbc'/>
[t03] <d s:KeyInfoxmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
[t04] <ds: RetrievalMethodURI='#EK'
           Escriba="http://www.w3.org/2001/04/xmlenc#EncryptedKey"/>
[t05] <ds:KeyName>Sally Doe</ds:KeyName>
[t06] </ds:KeyInfo>
[t07] <CipherData><CipherValue>DEADBEEF</CipherValue></CipherData>
[t08] </EncryptedData>
```

`[t02]`Este (AES-128-CBC) es un cifrado de clave simétrica.

`[t04] ds:RetrievalMethod`se utiliza para indicar la ubicación de una clave con tipo `&xenc;EncryptedKey`. La clave (AES) se encuentra en '#EK'.

`[t05] ds:KeyName`proporciona un método alternativo para identificar la clave necesaria para descifrar el archivo `CipherData`. Cualquiera o ambos `ds:KeyName`y `ds:KeyRetrievalMethod` podrían usarse para identificar la misma clave.

Dentro del mismo documento XML, existía una `EncryptedKey` estructura a la que se hacía referencia en `[t04]`:

```
[t09] <Id. de clave cifrada='EK' xmlns='http://www.w3.org/2001/04/xmlenc#'>
[t10] <Método de cifrado
         Algoritmo="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
[t11] <ds:KeyInfo xmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
[t12] <ds:KeyName>John Smith</ ds:KeyName>
[t13] </ds:KeyInfo>
[t14] <CipherData><CipherValue>xyzabc</CipherValue></CipherData>
[t15] <Lista de referencias>
[t16] <URI de referencia de datos='#ED'/>
[t17] </ListaReferencia>
[t18] <CarriedKeyName>Sally Doe</CarriedKeyName>
[t19] </EncryptedKey>
```

`[t09]`El `EncryptedKey`elemento es similar al `EncryptedData`elemento excepto que los datos cifrados son siempre un valor clave.

`[t10]`Es `EncryptionMethod`el algoritmo de clave pública RSA.

`[t12]` `ds:KeyName`de "John Smith" es una propiedad de la clave necesaria para descifrar (usando RSA) el archivo `CipherData`.

`[t14]`El `CipherData`'s `CipherValue` es una secuencia de octetos que es procesada (serializada, cifrada y codificada) por un objeto cifrado de referencia `EncryptionMethod`. (Tenga en cuenta que una EncryptedKey `EncryptionMethod`es el algoritmo utilizado para cifrar estos octetos y no habla de qué tipo de octetos son).

`[t15-17]`A `ReferenceList`identifica los objetos cifrados (`DataReference`y `KeyReference`) cifrados con esta clave. Contiene `ReferenceList`una lista de referencias a datos cifrados por la clave simétrica contenida dentro de esta estructura.

`[t18]`El `CarriedKeyName`elemento se utiliza para identificar el valor de la clave cifrada al que puede hacer referencia el `KeyName`elemento en `ds:KeyInfo`. (Dado que los valores de los atributos de ID deben ser exclusivos de un documento, `CarriedKeyName`puede indicar que varias `EncryptedKey`estructuras contienen el mismo valor de clave cifrado para diferentes destinatarios).

3 Sintaxis de cifrado

Esta sección proporciona una descripción detallada de la sintaxis y las funciones del cifrado XML. Las características descritas en esta sección DEBEN implementarse a menos que se indique lo contrario. La sintaxis se define mediante [XML-Schema] con el siguiente preámbulo XML, declaración, entidad interna e importación:

```
Definición del esquema:

<?xml versión="1.0" codificación="utf-8"?>
<!DOCTYPE esquema PUBLIC "-//W3C//DTD XMLSchema 200102//ES"
 "http://www.w3.org/2001/XMLSchema.dtd"
 [
   <!ATTLIST esquema
     xmlns:xenc CDATA #FIXED 'http://www.w3.org/2001/04/xmlenc#'
     xmlns:ds CDATA #FIXED 'http://www.w3.org/2000/09/xmldsig#'>
   <!ENTITY xenc 'http://www.w3.org/2001/04/xmlenc#'>
   <!ENTIDAD % p ''>
   <!ENTIDAD % s ''>
  ]>

<esquema xmlns='http://www.w3.org/2001/XMLSchema' versión='1.0'
        xmlns:ds='http://www.w3.org/2000/09/xmldsig#'
        xmlns:xenc='http://www.w3.org/2001/04/xmlenc#'
        targetNamespace='http://www.w3.org/2001/04/xmlenc#'
        elemento FormDefault='calificado'>

  <importar espacio de nombres='http://www.w3.org/2000/09/xmldsig#'
          esquemaLocation='http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/xmldsig-core-schema.xsd'/>
```

3.1 El `EncryptedType`elemento

`EncryptedType` is the abstract type from which `EncryptedData` and `EncryptedKey` are derived. While these two latter element types are very similar with respect to their content models, a syntactical distinction is useful to processing. Implementation MUST generate laxly schema valid [XML-schema] `EncryptedData` or `EncryptedKey` as specified by the subsequent schema declarations. (Note the laxly schema valid generation means that the content permitted by `xsd:ANY` need not be valid.) Implementations SHOULD create these XML structures (`EncryptedType` elements and their descendents/content) in Normalization Form C [NFC, NFC-Corrigendum].

```
Schema Definition:

<complexType name='EncryptedType' abstract='true'>
  <sequence>
    <element name='EncryptionMethod' type='xenc:EncryptionMethodType'
             minOccurs='0'/>
    <element ref='ds:KeyInfo' minOccurs='0'/>
    <element ref='xenc:CipherData'/>
    <element ref='xenc:EncryptionProperties' minOccurs='0'/>
  </sequence>
  <attribute name='Id' type='ID' use='optional'/>
  <attribute name='Type' type='anyURI' use='optional'/>
  <attribute name='MimeType' type='string' use='optional'/>
  <attribute name='Encoding' type='anyURI' use='optional'/>
 </complexType>
```

`EncryptionMethod` is an optional element that describes the encryption algorithm applied to the cipher data. If the element is absent, the encryption algorithm must be known by the recipient or the decryption will fail.

`ds:KeyInfo` is an optional element, defined by [XML-DSIG], that carries information about the key used to encrypt the data. Subsequent sections of this specification define new elements that may appear as children of `ds:KeyInfo`.

`CipherData` is a mandatory element that contains the `CipherValue` or `CipherReference` with the encrypted data.

`EncryptionProperties` can contain additional information concerning the generation of the `EncryptedType` (e.g., date/time stamp).

`Id` is an optional attribute providing for the standard method of assigning a string id to the element within the document context.

`Type` is an optional attribute identifying type information about the plaintext form of the encrypted content. While optional, this specification takes advantage of it for mandatory processing described in *Processing Rules: Decryption* (section 4.2). If the `EncryptedData` element contains data of `Type` 'element' or element 'content', and replaces that data in an XML document context, it is strongly recommended the `Type` attribute be provided. Without this information, the decryptor will be unable to automatically restore the XML document to its original cleartext form.

`MimeType` is an optional (advisory) attribute which describes the media type of the data which has been encrypted. The value of this attribute is a string with values defined by [MIME]. For example, if the data that is encrypted is a base64 encoded PNG, the transfer `Encoding` may be specified as 'http://www.w3.org/2000/09/xmldsig#base64' and the `MimeType` as 'image/png'. This attribute is purely advisory; no validation of the `MimeType` information is required and it does not indicate the encryption application must do any additional processing. Note, this information may not be necessary if it is already bound to the identifier in the `Type` attribute. For example, the Element and Content types defined in this specification are always UTF-8 encoded text.

3.2 The EncryptionMethod Element

EncryptionMethod is an optional element that describes the encryption algorithm applied to the cipher data. If the element is absent, the encryption algorithm must be known by the recipient or the decryption will fail.

```
Schema Definition:

<complexType name='EncryptionMethodType' mixed='true'>
  <sequence>
    <element name='KeySize' minOccurs='0' type='xenc:KeySizeType'/>
    <element name='OAEPparams' minOccurs='0' type='base64Binary'/>
    <any namespace='##other' minOccurs='0' maxOccurs='unbounded'/>
  </sequence>
  <attribute name='Algorithm' type='anyURI' use='required'/>
</complexType>
```

The permitted child elements of the `EncryptionMethod` are determined by the specific value of the `Algorithm` attribute URI, and the `KeySize` child element is always permitted. For example, the RSA-OAEP algorithm (section 5.4.2) uses the `ds:DigestMethod` and `OAEPparams` elements. (We rely upon the `ANY` schema construct because it is not possible to specify element content based on the value of an attribute.)

The presence of any child element under `EncryptionMethod` which is not permitted by the algorithm or the presence of a `KeySize` child inconsistent with the algorithm MUST be treated as an error. (All algorithm URIs specified in this document imply a key size but this is not true in general. Most popular stream cipher algorithms take variable size keys.)

3.3 The `CipherData` Element

The `CipherData` is a mandatory element that provides the encrypted data. It must either contain the encrypted octet sequence as base64 encoded text of the `CipherValue` element, or provide a reference to an external location containing the encrypted octet sequence via the `CipherReference` element.

```
Schema Definition:

<element name='CipherData' type='xenc:CipherDataType'/>
<complexType name='CipherDataType'>
   <choice>
     <element name='CipherValue' type='base64Binary'/>
     <element ref='xenc:CipherReference'/>
   </choice>
 </complexType>
```

3.3.1 The `CipherReference` Element

If `CipherValue` is not supplied directly, the `CipherReference` identifies a source which, when processed, yields the encrypted octet sequence.

The actual value is obtained as follows. The `CipherReference URI` contains an identifier that is dereferenced. Should the `CipherReference` element contain an OPTIONAL sequence of `Transform`s, the data resulting from dereferencing the URI is transformed as specified so as to yield the intended cipher value. For example, if the value is base64 encoded within an XML document; the transforms could specify an XPath expression followed by a base64 decoding so as to extract the octets.

The syntax of the `URI` and `Transforms` is similar to that of [XML-DSIG]. However, there is a difference between signature and encryption processing. In [XML-DSIG] both generation and validation processing start with the same source data and perform that transform in the same order. In encryption, the decryptor has only the cipher data and the specified transforms are enumerated for the decryptor, in the order necessary to obtain the octets. Consequently, because it has different semantics `Transforms` is in the `&xenc;` namespace.

For example, if the relevant cipher value is captured within a `CipherValue` element within a different XML document, the `CipherReference` might look as follows:

```
<CipherReference URI="http://www.example.com/CipherValues.xml">
  <Transforms>
    <ds:Transform
     Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
        <ds:XPath xmlns:rep="http://www.example.org/repository">
          self::text()[parent::rep:CipherValue[@Id="example1"]]
        </ds:XPath>
    </ds:Transform>
    <ds:Algoritmo de transformación="http://www.w3.org/2000/09/xmldsig#base64"/>
  </Transforma>
</CipherReference>
```

Las implementaciones DEBEN admitir la `CipherReference`función y la misma codificación URI, desreferenciación, esquema y códigos de respuesta HTTP que los de [XML-DSIG]. La `Transform`característica y los algoritmos de transformación particulares son OPCIONALES.

```
Definición del esquema:

<elemento nombre=' CipherReference' tipo=' xenc:CipherReferenceType'/>
 <nombre de tipo complejo=' CipherReferenceType'>
     <secuencia>
       <elemento nombre='Transforms' tipo='xenc:TransformsType' minOccurs='0'/>
     </secuencia>
     <nombre de atributo='URI' tipo='cualquierURI' uso='requerido'/>
 </tipocomplejo>

  <nombre de tipo complejo='Tipo de transformación'>
     <secuencia>
       <elemento ref='ds:Transform' maxOccurs='ilimitado'/>
     </secuencia>
   </tipocomplejo>
```

3.4 El `EncryptedData`elemento

El `EncryptedData`elemento es el elemento central de la sintaxis. Su `CipherData`elemento secundario no solo contiene los datos cifrados, sino que también es el elemento que reemplaza al elemento cifrado o sirve como raíz del nuevo documento.

```
Definición del esquema:

<elemento nombre=' EncryptedData' tipo=' xenc:EncryptedDataType'/>
<nombre de tipo complejo=' EncryptedDataType'>
  <Contenido complejo>
   <base de extensión=' xenc:EncryptedType'>
   </extensión>
  </complexContent>
</tipocomplejo>
```

3.5 Extensiones al`ds:KeyInfo` elemento

Hay tres formas de `CipherData`proporcionar el material de claves necesario para descifrar:

1. El elemento `EncryptedData`o `EncryptedKey`especifica el material de clave asociado a través de un elemento secundario de `ds:KeyInfo`. Todos los elementos secundarios de ds: `KeyInfo`especificados en [XML-DSIG] PUEDEN usarse como calificados:
   1. La compatibilidad con `ds:KeyValue`es OPCIONAL y puede usarse para transportar claves públicas, como los valores clave Diffie-Hellman (sección 5.5.1). (Obviamente NO SE RECOMIENDA incluir la clave de descifrado de texto plano, ya sea una clave privada o secreta).
   2. Se RECOMIENDA el soporte de `ds:KeyName`para hacer referencia a un `.EncryptedKey CarriedKeyName`
   3. `ds:RetrievalMethod`Se REQUIERE soporte para el mismo documento .

   Además, proporcionamos dos elementos secundarios adicionales: las aplicaciones DEBEN ser compatibles `EncryptedKey` (sección 3.5.1) y PUEDEN ser compatibles `AgreementMethod`(sección 5.5).
2. Un elemento separado (no dentro de `ds:KeyInfo`) `EncryptedKey`puede especificar el `EncryptedData`o `EncryptedKey`al cual se aplicará su clave descifrada a través de `DataReference`o `KeyReference`(sección 3.6).
3. El material de claves lo puede determinar el destinatario según el contexto de la aplicación y, por lo tanto, no es necesario mencionarlo explícitamente en el XML transmitido.

3.5.1 El `EncryptedKey`elemento

Identificador
`Type="http://www.w3.org/2001/04/xmlenc#EncryptedKey"`

(Esto se puede usar dentro de un `ds:RetrievalMethod`elemento para identificar el tipo de referente).

El `EncryptedKey`elemento se utiliza para transportar claves de cifrado desde el origen hasta uno o varios destinatarios conocidos. Puede usarse como un documento XML independiente, colocarse dentro de un documento de aplicación o aparecer dentro de un `EncryptedData`elemento como hijo de un `ds:KeyInfo` elemento. El valor de la clave siempre está cifrado para los destinatarios. Cuando `EncryptedKey`se descifra, los octetos resultantes se ponen a disposición del `EncryptionMethod`algoritmo sin ningún procesamiento adicional.

```
Definición del esquema:

<elemento nombre=' EncryptedKey' tipo=' xenc:EncryptedKeyType'/>
<nombre de tipo complejo=' EncryptedKeyType'>
  <Contenido complejo>
    <base de extensión=' xenc:EncryptedType'>
      <secuencia>
        <elemento ref=' xenc:ReferenceList' minOccurs='0'/>
        <elemento nombre=' CarriedKeyName' tipo='cadena' minOccurs='0'/>
      </secuencia>
      <nombre del atributo='Destinatario' tipo='cadena' uso='opcional'/>
    </extensión>
  </complexContent>
</tipocomplejo>
```

`ReferenceList`es un elemento opcional que contiene punteros a datos y claves cifradas con esta clave. La lista de referencias puede contener múltiples referencias `EncryptedKey`y `EncryptedData`elementos. Esto se hace usando elementos `KeyReference`y `DataReference`respectivamente. Estos se definen a continuación.

`CarriedKeyName`es un elemento opcional para asociar un nombre legible por el usuario con el valor clave. Esto luego se puede usar para hacer referencia a la clave usando el `ds:KeyName`elemento dentro de `ds:KeyInfo`. La misma `CarriedKeyName`etiqueta, a diferencia de un tipo de identificación, puede aparecer varias veces dentro de un solo documento. El valor de la clave debe ser el mismo en todos

EncryptedKeylos elementos identificados con la misma CarriedKeyNameetiqueta dentro de un único documento XML. Tenga en cuenta que debido a que los espacios en blanco son significativos en el valor del ds:KeyName elemento, los espacios en blanco también son significativos en el valor del CarriedKeyNameelemento.

Recipientes un atributo opcional que contiene una pista sobre a qué destinatario está destinado este valor de clave cifrada. Su contenido depende de la aplicación.

El Typeatributo heredado de EncryptedType se puede utilizar para especificar aún más el tipo de clave cifrada si EncryptionMethod Algorithmno define una codificación/representación inequívoca. (Tenga en cuenta que todos los algoritmos de esta especificación tienen una representación inequívoca para sus estructuras clave asociadas).

3.5.2 El ds:RetrievalMethodelemento

El con a de ' ' proporciona una manera de expresar un enlace a un elemento que contiene la clave necesaria para descifrar el elemento asociado con un o . El de este tipo es siempre hijo del elemento y puede aparecer varias veces. Si hay más de una instancia de a de este tipo, entonces los objetos a los que se hace referencia deben contener el mismo valor de clave, posiblemente cifrado de diferentes maneras o para diferentes destinatarios.ds:RetrievalMethod [XML-DSIG]Typehttp://www.w3.org/2001/04/xmlenc#EncryptedKeyEncryptedKeyCipherDataEncryptedDataEncryptedKeyds:RetrievalMethodds:KeyInfo

```
Definición del esquema:

<!--
    <nombre del atributo='Tipo' tipo='cualquierURI' uso='opcional'
    fijo='http://www.w3.org/2001/04/xmlenc# EncryptedKey' />
-->
```

3.6 El ReferenceListelemento

ReferenceListes un elemento que contiene punteros desde un valor clave de an EncryptedKeya elementos cifrados por ese valor clave (EncryptedDatao EncryptedKeyelementos).

```
Definición del esquema:

<nombre del elemento='Lista de referencias'>
  <tipocomplejo>
    <elección minOccurs='1' maxOccurs='ilimitado'>
      <elemento nombre='DataReference' tipo='xenc:ReferenceType'/>
      <elemento nombre='KeyReference' tipo='xenc:ReferenceType'/>
    </elección>
  </tipocomplejo>
</elemento>

<nombre de tipo complejo=' ReferenceType'>
  <secuencia>
    <cualquier espacio de nombres='##other' minOccurs='0' maxOccurs='ilimitado'/>
  </secuencia>
  <nombre de atributo='URI' tipo='cualquierURI' uso='requerido'/>
</tipocomplejo>
```

DataReferenceLos elementos se utilizan para referirse a EncryptedDataelementos que se cifraron utilizando la clave definida en el EncryptedKeyelemento adjunto. Pueden aparecer varios DataReferenceelementos si EncryptedDataexisten varios elementos cifrados con la misma clave.

KeyReferenceLos elementos se utilizan para referirse a EncryptedKeyelementos que se cifraron utilizando la clave definida en el EncryptedKeyelemento adjunto. Pueden aparecer varios KeyReferenceelementos si EncryptedKeyexisten varios elementos cifrados con la misma clave.

Para ambos tipos de referencias, se pueden especificar opcionalmente elementos secundarios para ayudar al destinatario a recuperar los elementos EncryptedKeyy/o EncryptedData. Estos podrían incluir información como transformaciones XPath, transformaciones de descompresión o información sobre cómo recuperar los elementos de una instalación de almacenamiento de documentos. Por ejemplo:

```
<Lista de referencia>
  <URI de referencia de datos="#factura34">
    <ds:Transformaciones>
      <ds:Algoritmo de transformación="http://www.w3.org/TR/1999/REC-xpath-19991116">
        <ds:XPath xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
            self::xenc:EncryptedData[@Id="ejemplo1"]
        </ds:XPath>
      </ds:Transformar>
    </ds:Transformaciones>
  </ReferenciaDeDatos>
</ReferenciaLista>
```

3.7 El EncryptionProperties elemento

Identificador
Type="http://www.w3.org/2001/04/xmlenc#EncryptionProperties"

(Esto se puede usar dentro de un ds:Referenceelemento para identificar el tipo de referente).

Se pueden colocar elementos de información adicional relacionados con la generación de EncryptedDatao en un elemento (por ejemplo, marca de fecha/hora o el número de serie del hardware criptográfico utilizado durante el cifrado). El atributo identifica la estructura que se describe. permite la inclusión de atributos del espacio de nombres XML que se incluirán (es decir, , y).EncryptedKeyEncryptionPropertyTargetEncryptedTypeanyAttributexml:spacexml:langxml:base

```
Definición del esquema:

<elemento nombre='EncryptionProperties' tipo='xenc:EncryptionPropertiesType'/>
<complexType nombre='EncryptionPropertiesType'>
```

```
  <secuencia>
    <elemento ref='xenc:EncryptionProperty' maxOccurs='ilimitado'/>
  </secuencia>
  <nombre del atributo='Id' tipo='ID' uso='opcional'/>
</tipocomplejo>

  <elemento nombre='EncryptionProperty' tipo='xenc:EncryptionPropertyType'/>
  <complexType nombre='EncryptionPropertyType' mixto='verdadero'>
    <elección maxOccurs='ilimitado'>
      <cualquier espacio de nombres='##other' ProcessContents='lax'/>
    </elección>
    <nombre del atributo='Destino' tipo='cualquierURI' uso='opcional'/>
    <nombre del atributo='Id' tipo='ID' uso='opcional'/>
    <anyAttribute namespace="http://www.w3.org/XML/1998/namespace"/>
  </tipocomplejo>
```

4 reglas de procesamiento

Esta sección describe las operaciones que se realizarán como parte del procesamiento de cifrado y descifrado mediante implementaciones de esta especificación. Los requisitos de conformidad se especifican en los siguientes roles:

Solicitud
: La aplicación que solicita la implementación de un cifrado XML mediante el suministro de datos y parámetros necesarios para su procesamiento.

cifrador
: Una implementación de cifrado XML con la función de cifrar datos.

Descifrador
: Una implementación de cifrado XML con la función de descifrar datos.

4.1 Cifrado

Para que cada elemento de datos se cifre como `EncryptedData`o `EncryptedKey`(elementos derivados de `EncryptedType`), el **cifrador** debe:

1. Seleccione el algoritmo (y los parámetros) que se utilizarán para cifrar estos datos.
2. Obtener y (opcionalmente) representar la clave.
   1. Si la clave debe identificarse (mediante nombre, URI o incluirse en un elemento secundario), construya la clave `ds:KeyInfo`según corresponda (p. ej. `ds:KeyName`, `ds:KeyValue`, `ds:RetrievalMethod`, etc.)
   2. Si se va a cifrar la clave en sí, construya un `EncryptedKey`elemento aplicando recursivamente este proceso de cifrado. El resultado puede ser hijo de `ds:KeyInfo`o puede existir en otro lugar y puede identificarse en el paso anterior.
3. cifrar los datos
   1. Si los datos son un ' elemento ' [XML , sección 3] o un elemento ' contenido ' [XML , sección 3.1], obtenga los octetos serializando los datos en UTF-8 como se especifica en [XML]. (La aplicación DEBE proporcionar datos XML en [NFC].) La serialización PUEDE ser realizada por el **cifrador** . Si el **cifrador** no se serializa, entonces la **aplicación** DEBE realizar la serialización.
   2. Si los datos son de cualquier otro tipo que aún no sean octetos, la **aplicación** DEBE serializarlos como octetos.
   3. Cifre los octetos utilizando el algoritmo y la clave de los pasos 1 y 2.
   4. A menos que el **descifrador** conozca implícitamente el tipo de datos cifrados, el **cifrado** DEBE proporcionar el tipo para la representación.

      La definición de este tipo como vinculado a un identificador especifica cómo obtener e interpretar los octetos de texto plano después del descifrado. Por ejemplo, el identificador podría indicar que los datos son una instancia de otra aplicación (por ejemplo, alguna aplicación de compresión XML) que debe procesarse más. O, si los datos son una secuencia de octetos simple, PUEDEN describirse con los atributos `MimeType`y `Encoding`. Por ejemplo, los datos pueden ser un documento XML (`MimeType="text/xml"`), una secuencia de caracteres (`MimeType="text/plain"`) o datos de imagen binaria (`MimeType="image/png"`).
4. Construya la estructura `EncryptedType`(`EncryptedData`o `EncryptedKey`):

   Una `EncryptedType`estructura representa toda la información analizada anteriormente, incluido el tipo de datos cifrados, el algoritmo de cifrado, los parámetros, la clave, el tipo de datos cifrados, etc.

   1. Si la secuencia de octetos cifrada obtenida en el paso 3 se va a almacenar en el `CipherData`elemento dentro de `EncryptedType`, entonces la secuencia de octetos cifrada se codifica en base64 y se inserta como contenido de un `CipherValue`elemento.
   2. Si la secuencia de octetos cifrada se va a almacenar externamente a la `EncryptedType`estructura, almacene o devuelva la secuencia de octetos cifrada y represente el URI y las transformaciones (si las hay) necesarias para que el descifrador recupere la secuencia de octetos cifrada dentro de un `CipherReference`elemento.
5. Procesar datos cifrados
   1. Si los `Type`datos cifrados son " elemento " o elemento " contenido ", entonces el **cifrado** DEBE poder devolver el `EncryptedData`elemento a la **aplicación** . La **aplicación** PUEDE utilizar esto como elemento de nivel superior en un nuevo documento XML o insertarlo en otro documento XML, lo que puede requerir una nueva codificación.

      El **cifrado** DEBE poder reemplazar el 'elemento' o 'contenido' no cifrado con el elemento EncryptedData. Cuando una **aplicación** requiere que se reemplace un elemento o contenido XML, proporciona el contexto del documento XML además de identificar el elemento o contenido que se va a reemplazar. El **cifrador** elimina el elemento o contenido identificado y lo inserta `EncryptedData`en su lugar.

      (Nota: si el `Type`"contenido" es el documento resultante del descifrado no estará bien formado si (a) el texto sin formato original no estaba bien formado (por ejemplo, PCDATA por sí solo no está bien formado) y (b) el `EncryptedData`elemento era anteriormente el elemento raíz del documento)
   2. Si el elemento `Type`de los datos cifrados *no es* " elemento " o " contenido " del elemento, entonces el **cifrador** siempre DEBE devolver el `EncryptedData`elemento a la **aplicación** . La **aplicación** PUEDE utilizar esto como elemento de nivel superior en un nuevo documento XML o insertarlo en otro documento XML, lo que puede requerir una nueva codificación.

4.2 Descifrado

Para que cada `EncryptedType`elemento derivado (es decir, `EncryptedData`o `EncryptedKey`) se descifre, el **descifrador** debe:

1. Procesar el elemento para determinar el algoritmo, parámetros y `ds:KeyInfo`elemento a utilizar. Si se omite alguna información, la **aplicación** DEBE proporcionarla.
2. Localice la clave de cifrado de datos según el `ds:KeyInfo` elemento, que puede contener uno o más elementos secundarios. Estos niños no tienen ningún orden de procesamiento implícito. Si la clave de cifrado de datos está cifrada, busque la clave correspondiente para descifrarla. (Este puede ser un paso recursivo ya que la clave de cifrado de clave puede estar cifrada). O bien, se puede recuperar la clave de cifrado de datos de un almacén local utilizando los atributos proporcionados o el enlace implícito.
3. Descifre los datos contenidos en el `CipherData`elemento.
    1. Si hay un `CipherValue`elemento secundario presente, entonces se recupera el valor de texto asociado y se decodifica en base64 para obtener la secuencia de octetos cifrada.
    2. Si hay un `CipherReference`elemento secundario presente, el URI y las transformaciones (si las hay) se utilizan para recuperar la secuencia de octetos cifrados.
    3. La secuencia de octetos cifrada se descifra utilizando el algoritmo/parámetros y el valor de clave ya determinados en los pasos 1 y 2.
4. Procesar datos descifrados de `Type`' elemento ' o elemento ' contenido '.
    1. La secuencia de octetos de texto sin cifrar obtenida en el paso 3 se interpreta como datos de caracteres codificados en UTF-8.
    2. El **descifrador** DEBE poder devolver el valor `Type`y los datos de caracteres XML codificados en UTF-8. NO ES NECESARIO que el **descifrador** realice la validación en el XML serializado.
    3. El **descifrador** DEBE admitir la capacidad de reemplazar el `EncryptedData`elemento con el ' elemento ' o el elemento ' contenido ' descifrado representado por los caracteres codificados en UTF-8. NO ES NECESARIO que el **descifrador** realice la validación del resultado de esta operación de reemplazo.

       La aplicación proporciona el contexto del documento XML e identifica el `EncryptedData`elemento que se reemplaza. Si el documento en el que se realiza el reemplazo no es UTF-8, el **descifrador** DEBE transcodificar los caracteres codificados en UTF-8 a la codificación de destino.
5. Procesar datos descifrados si `Type`no están especificados o *no* son ' elemento ' o elemento ' contenido '.
    1. La secuencia de octetos de texto sin cifrar obtenida en **el paso 3** DEBE devolverse a la **aplicación** para su posterior procesamiento junto con los valores de atributo, y cuando se `Type`especifiquen `MimeType`. y son asesores. El valor es normativo ya que puede contener información necesaria para el procesamiento o interpretación de los datos por parte de la aplicación.`EncodingMimeTypeEncodingType`
    2. Tenga en cuenta que este paso incluye el procesamiento de datos descifrados de un archivo `EncryptedKey`. La secuencia de octetos de texto sin cifrar representa un valor clave y la aplicación la utiliza para descifrar otros `EncryptedType`elementos.

4.3 Cifrado XML

Las operaciones de cifrado y descifrado se transforman en octetos. La **aplicación** es responsable de ordenar el XML de manera que pueda serializarse en una secuencia de octetos, cifrarse, descifrarse y ser de utilidad para el destinatario.

Por ejemplo, si la aplicación desea canonicalizar sus datos o codificar/comprimir los datos en un formato de empaquetado XML, la aplicación necesita ordenar el XML en consecuencia e identificar el tipo resultante mediante el `EncryptedData Type`atributo. La probabilidad de que el descifrado y el procesamiento posterior sean exitosos dependerán del soporte del destinatario para el tipo dado. Además, si se pretende que los datos se procesen antes del cifrado y después del descifrado (por ejemplo, validación de firma XML [XML-DSIG] o una transformación XSLT), la aplicación de cifrado debe tener cuidado de preservar la información necesaria para el éxito de ese proceso.

Para fines de interoperabilidad, DEBEN implementarse los siguientes tipos de modo que una implementación pueda tomar como entrada y producir como salida datos que coincidan con las reglas de producción 39 y 43 de [XML] :

elemento ' http://www.w3.org/2001/04/xmlenc#Element '
"[39] elemento ::= `EmptyElemTag` | Contenido de STag ETag "
contenido ' http://www.w3.org/2001/04/xmlenc#Content '
"[43] contenido ::= `CharData`? ((elemento | Referencia | CDSect | PI | Comentario) `CharData`?)*"

Las siguientes secciones contienen especificaciones para descifrar, reemplazar y serializar contenido XML (es decir, `Type`' elemento ' o elemento ' contenido ') utilizando el modelo de datos [XPath]. Estas secciones no son normativas y son OPCIONALES para los implementadores de esta especificación, pero pueden ser referenciadas normativamente y OBLIGATORIAS para otras especificaciones que requieren un procesamiento consistente para aplicaciones, como [XML-DSIG-Decrypt] .

4.3.1 Una implementación de Decrypt (no normativa)

Donde *P* es el contexto en el que se debe analizar el XML serializado (un nodo de documento o nodo de elemento) y *O* es la secuencia de octetos que representa caracteres codificados en UTF-8 resultantes del paso 4.3 en el *Procesamiento de descifrado* (sección 4.2). *Y* es un conjunto de nodos que representa el contenido descifrado obtenido mediante los siguientes pasos:

1. Sea *C* el **contexto de análisis** de un hijo de *P* , que consta de los siguientes elementos:
    - Prefijo y nombre del espacio de nombres de cada espacio de nombres que está dentro del alcance de *P* .
    - Nombre y valor de cada entidad general que es efectiva para el documento XML que causa *P* .
2. Envuelva el flujo de octetos descifrado *O* en el contexto *C* como se especifica en *Ajuste de texto* .
3. Analice el flujo de octetos envueltos como se describe en *El modelo de procesamiento de referencia* (sección 4.3.3.2) de [Firma XML], lo que da como resultado un conjunto de nodos.
4. *Y* es el conjunto de nodos obtenido al eliminar el nodo raíz, el nodo del elemento envolvente y su conjunto asociado de nodos de atributos y espacios de nombres del conjunto de nodos obtenido en el Paso 3.

4.3.2 Una implementación de descifrado y reemplazo (no normativa)

Donde *X* es el conjunto de nodos [XPath] correspondiente a un documento XML y *e* es un `EncryptedData`nodo de elemento en *X* .

1. *Z* es un conjunto de nodos [XPath] idéntico a X excepto donde el nodo de elemento *e* es un `EncryptedData`tipo de elemento. En ese caso:
    1. Descifre *e* en el contexto de su nodo principal como se especifica en la *Implementación de descifrado* (sección 4.3.1), lo que produce *Y* , un conjunto de nodos [XPath].

2. Incluya *Y* en lugar de *e* y sus descendientes en *X* . Dado que [XPath] no define métodos para reemplazar conjuntos de nodos de diferentes documentos, el resultado DEBE ser equivalente a reemplazar e con el flujo de octetos resultante de su descifrado en la forma serializada de X y *analizar* el documento. Sin embargo, el método real para realizar esta operación queda en manos del implementador.

4.3.3 Serialización de XML (no normativo)

Consideraciones sobre el espacio de nombres predeterminado

En *Cifrado de XML* (sección 4.1, paso 3.1), al serializar un fragmento XML DEBE tenerse especial cuidado con respecto a los espacios de nombres predeterminados. Si los datos se descifran posteriormente en el contexto de un documento XML principal, la serialización puede producir elementos en el espacio de nombres incorrecto. Considere el siguiente fragmento de XML:

```
<Documento xmlns="http://ejemplo.org/">
  <ToBeEncrypted xmlns="" />
</Documento>
```

La serialización del `ToBeEncrypted`fragmento de elemento mediante [XML-C14N] daría como resultado los caracteres " `<ToBeEncrypted></ToBeEncrypted>`" como una secuencia de octetos. El documento cifrado resultante sería:

```
<Documento xmlns="http://ejemplo.org/">
  <EncryptedData xmlns="...">
    <!-- Contiene el cifrado
         "<ToBeEncrypted></ToBeEncrypted>" -->
  </EncryptedData>
</Documento>
```

Descifrar y reemplazar el `EncryptedData`contenido de este documento produciría el siguiente resultado incorrecto:

```
<Documento xmlns="http://ejemplo.org/">
  <Para ser cifrado/>
</Documento>
```

Este problema surge porque la mayoría de las serializaciones XML asumen que los datos serializados se analizarán directamente en un contexto donde no existe una declaración de espacio de nombres predeterminada. En consecuencia, no declaran de forma redundante el espacio de nombres predeterminado vacío con una extensión `xmlns=""`. Sin embargo, si los datos serializados se analizan en un contexto donde una declaración de espacio de nombres predeterminada está dentro del alcance (por ejemplo, el contexto de análisis de una *implementación de A Decrypt* (sección 4.3.1)), entonces puede afectar la interpretación de los datos serializados.

Para resolver este problema, se PUEDE aumentar un algoritmo de canonicalización de la siguiente manera para usarlo como serializador de cifrado XML:

- `xmlns=""`Se DEBE emitir una declaración de espacio de nombres predeterminada con un valor vacío (es decir,) donde normalmente el algoritmo de canonicalización la suprimiría.

Si bien es posible que el resultado no esté en la forma canónica adecuada, esto es inofensivo ya que el flujo de octetos resultante no se utilizará directamente en un cálculo del valor de firma [Firma XML]. Volviendo al ejemplo anterior con nuestro nuevo aumento, el `ToBeEncrypted`elemento se serializaría de la siguiente manera:

```
<ToBeEncrypted xmlns=""></ToBeEncrypted>
```

Cuando se procesa en el contexto del documento principal, este fragmento serializado se analizará e interpretará correctamente.

Este aumento se puede aplicar retroactivamente a una implementación de canonicalización existente canonicalizando cada nodo vértice y sus descendientes del conjunto de nodos, insertándolos `xmlns=""`en los puntos apropiados y concatenando los flujos de octetos resultantes.

Consideraciones sobre atributos XML

Se debe prestar una atención similar entre la relación de un fragmento y el contexto en el que se inserta a los atributos `xml:base`, `xml:lang`y `xml:space`como se menciona en las *Consideraciones de seguridad* de [XML-exc-C14N]. Por ejemplo, si el elemento:

```
<Bongo href="ejemplo.xml"/>
```

se toma de un contexto y se serializa sin atributo `xml:base`[XML-Base] y se analiza en el contexto del elemento:

```
<Baz xml:base="http://ejemplo.org/"/>
```

el resultado será:

```
<Baz xml:base="http://example.org/"><Bongo href="ejemplo.xml"/></Baz>
```

`Bongo`'s `href`se interpreta posteriormente como " `http://example.org/example.xml`". Si este no es el URI correcto, `Bongo`debería haberse serializado con su propio `xml:base`atributo.

Desafortunadamente, la recomendación de que se emita un valor vacío para separar el espacio de nombres predeterminado del fragmento del contexto en el que se inserta no se puede realizar para los atributos `xml:base`y `xml:space`. (*El error 41* de la errata de especificación XML 1.0 de segunda edición aclara que un valor de cadena vacío del atributo `xml:lang` *se* considera como si "no hubiera información de idioma disponible, como si `xml:lang`no se hubiera especificado".) La interpretación de un valor vacío para los atributos `xml:base`o `xml:space`no están definidos o mantienen el valor contextual. En consecuencia, las aplicaciones DEBEN garantizar (1) que los fragmentos que se van a cifrar no dependan de atributos XML, o (2) si son dependientes y se pretende que el documento resultante sea válido [XML], la definición del fragmento *permite* la presencia del atributos y que los atributos tengan valores no vacíos.

4.3.4 Ajuste de texto (no normativo)

Esta sección especifica el proceso para ajustar texto en un contexto de análisis determinado. El proceso se basa en la propuesta de Richard Tobin [Tobin] para construir el conjunto de información [XML-Infoset] de una entidad externa.

El proceso consta de los siguientes pasos:

1. Si el contexto de análisis contiene entidades generales, emita una declaración de tipo de documento que proporcione declaraciones de entidades.
2. Emita una `dummy`etiqueta de inicio de elemento con atributos de declaración de espacio de nombres que declaren todos los espacios de nombres en el contexto de análisis.
3. Emitir el texto.
4. Emitir una `dummy`etiqueta final de elemento.

En los pasos anteriores, la declaración del tipo de documento y `dummy` las etiquetas de elementos DEBEN codificarse en UTF-8.

Considere el siguiente documento que contiene un `EncryptedData` elemento:

```
<!DOCTYPE Documento [
  <!ENTITY dsig "http://www.w3.org/2000/09/xmldsig#">
]>
<Documento xmlns="http://ejemplo.org/">
  <foo:Cuerpo xmlns:foo="http://example.org/foo">
    <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
                   Escriba=" http://www.w3.org/2001/04/xmlenc#Element ">
      ...
    </EncryptedData>
  </foo:Cuerpo>
</Documento>
```

Si el `EncryptedData`elemento se alimenta y se descifra en el texto " `<One><foo:Two/></One>`", entonces el formato ajustado es el siguiente:

```
<!DOCTYPE ficticio [
  <!ENTITY dsig "http://www.w3.org/2000/09/xmldsig#">
]>
<dummy xmlns="http://ejemplo.org/"
       xmlns:foo="http://example.org/foo"><Uno><foo:Two/></Uno></dummy>
```

5. Algoritmos

This section discusses algorithms used with the XML Encryption specification. Entries contain the identifier to be used as the value of the `Algorithm` attribute of the `EncryptionMethod` element or other element representing the role of the algorithm, a reference to the formal specification, definitions for the representation of keys and the results of cryptographic operations where applicable, and general applicability comments.

5.1 Algorithm Identifiers and Implementation Requirements

All algorithms listed below have implicit parameters depending on their role. For example, the data to be encrypted or decrypted, keying material, and direction of operation (encrypting or decrypting) for encryption algorithms. Any explicit additional parameters to an algorithm appear as content elements within the element. Such parameter child elements have descriptive element names, which are frequently algorithm specific, and SHOULD be in the same namespace as this XML Encryption specification, the XML Signature specification, or in an algorithm specific namespace. An example of such an explicit parameter could be a nonce (unique quantity) provided to a key agreement algorithm.

This specification defines a set of algorithms, their URIs, and requirements for implementation. Levels of requirement specified, such as "REQUIRED" or "OPTIONAL", refere to implementation, not use. Furthermore, the mechanism is extensible, and alternative algorithms may be used.

Table of Algorithms

The table below lists the categories of algorithms. Within each category, a brief name, the level of implementation requirement, and an identifying URI are given for each algorithm.

Block Encryption

1. REQUIRED TRIPLEDES
   http://www.w3.org/2001/04/xmlenc#tripledes-cbc
2. REQUIRED AES-128
   http://www.w3.org/2001/04/xmlenc#aes128-cbc
3. REQUIRED AES-256
   http://www.w3.org/2001/04/xmlenc#aes256-cbc
4. OPTIONAL AES-192
   http://www.w3.org/2001/04/xmlenc#aes192-cbc

Stream Encryption

1. none
   Syntax and recommendations are given below to support user specified algorithms.

Key Transport

1. REQUIRED RSA-v1.5
   http://www.w3.org/2001/04/xmlenc#rsa-1_5
2. REQUIRED RSA-OAEP
   http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p

Key Agreement

1. OPTIONAL Diffie-Hellman
   http://www.w3.org/2001/04/xmlenc#dh

Symmetric Key Wrap

1. REQUIRED TRIPLEDES KeyWrap
   http://www.w3.org/2001/04/xmlenc#kw-tripledes
2. REQUIRED AES-128 KeyWrap
   http://www.w3.org/2001/04/xmlenc#kw-aes128
3. REQUIRED AES-256 KeyWrap
   http://www.w3.org/2001/04/xmlenc#kw-aes256
4. OPTIONAL AES-192 KeyWrap
   http://www.w3.org/2001/04/xmlenc#kw-aes192

Message Digest

1. REQUIRED SHA1
   http://www.w3.org/2000/09/xmldsig#sha1
2. RECOMENDADO SHA256
   http://www.w3.org/2001/04/xmlenc#sha256
3. SHA512 OPCIONAL
   http://www.w3.org/2001/04/xmlenc#sha512
4. RIPEMD-160 OPCIONAL
   http://www.w3.org/2001/04/xmlenc#ripemd160

Autenticación de mensajes

1. Firma digital XML RECOMENDADA
   http://www.w3.org/2000/09/xmldsig#

Canonicalización

1. XML canónico OPCIONAL (omite comentarios)
   http://www.w3.org/TR/2001/REC-xml-c14n-20010315
2. XML canónico OPCIONAL con comentarios
   http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments
3. OPCIONAL Canonicalización XML exclusiva (omite comentarios)
   http://www.w3.org/2001/10/xml-exc-c14n#
4. OPCIONAL Canonicalización XML exclusiva con comentarios
   http://www.w3.org/2001/10/xml-exc-c14n#WithComments

Codificación

1. REQUERIDO base64
   http://www.w3.org/2000/09/xmldsig#base64

5.2 Algoritmos de cifrado de bloques

Los algoritmos de cifrado de bloques están diseñados para cifrar y descifrar datos en bloques de varios octetos de tamaño fijo. Sus identificadores aparecen como el valor de los `Algorithm`atributos de `EncryptionMethod`los elementos hijos de `EncryptedData`.

Los algoritmos de cifrado de bloques toman, como argumentos implícitos, los datos que se van a cifrar o descifrar, el material de clave y su dirección de operación. Para todos estos algoritmos especificados a continuación, se requiere un vector de inicialización (IV) codificado con el texto cifrado. Para los algoritmos de cifrado de bloques especificados por el usuario, el IV, si lo hubiera, podría especificarse junto con los datos cifrados, como un elemento de contenido del algoritmo o en cualquier otro lugar.

El IV está codificado con y antes del texto cifrado para los algoritmos siguientes para facilitar la disponibilidad del código de descifrado y enfatizar su asociación con el texto cifrado. Las buenas prácticas criptográficas requieren que se utilice un IV diferente para cada cifrado.

Relleno

Dado que los datos que se cifran son un número arbitrario de octetos, es posible que no sean un múltiplo del tamaño del bloque. Esto se resuelve rellenando el texto sin formato hasta el tamaño del bloque antes del cifrado y deshaciendo el relleno después del descifrado. El algoritmo de relleno consiste en calcular el número de octetos más pequeño distinto de cero, por ejemplo `N`, que debe añadirse al texto sin formato para que sea un múltiplo del tamaño del bloque. Supondremos que el tamaño del bloque es `B` de octetos, por lo que `N`está en el rango de 1 a `B`. Rellene añadiendo al texto sin formato un sufijo de `N-1`bytes de relleno arbitrarios y un byte final cuyo valor sea `N`. Al descifrar, simplemente tome el último byte y, después de verificarlo, elimine esa cantidad de bytes del final del texto cifrado descifrado.

Por ejemplo, supongamos un tamaño de bloque de 8 bytes y un texto sin formato de `0x616263`. El texto sin formato acolchado estaría entonces `0x616263??????????05`donde "??" Los bytes pueden tener cualquier valor. De manera similar, el texto sin formato de `0x2122232425262728`se rellenaría con `0x2122232425262728??????????????08`.

5.2.1 Triple DES

Identificador:
http://www.w3.org/2001/04/xmlenc#tripledes-cbc (REQUERIDO)

ANSI X9.52 [TRIPLEDES] especifica tres operaciones FIPS 46-3 [DES] secuenciales. El cifrado XML TRIPLEDES consta de un cifrado DES, un descifrado DES y un cifrado DES utilizado en el modo Cipher Block Chaining (CBC) con 192 bits de clave y un vector de inicialización (IV) de 64 bits. De los bits clave, los primeros 64 bits se utilizan en la primera operación DES, los segundos 64 bits en la operación DES intermedia y los terceros 64 bits en la última operación DES.

Nota: Cada uno de estos 64 bits de clave contiene 56 bits efectivos y 8 bits de paridad. Por tanto, sólo hay 168 bits operativos de los 192 que se transportan para una clave TRIPLEDES. (Dependiendo del criterio utilizado para el análisis, se puede pensar que la fuerza efectiva de la clave es de 112 bits (debido a los ataques intermedios) o incluso menos).

El texto cifrado resultante tiene el prefijo IV. Si se incluye en la salida XML, está codificado en base64. Un ejemplo de método de cifrado TRIPLEDES es el siguiente:

```
<Método de cifrado
 Algoritmo="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
```

5.2.2 AES

Identificador:
http://www.w3.org/2001/04/xmlenc#aes128-cbc (REQUERIDO)
http://www.w3.org/2001/04/xmlenc#aes192-cbc (OPCIONAL)
http://www.w3.org/2001/04/xmlenc#aes256-cbc (REQUERIDO)

[AES] se utiliza en el modo Cipher Block Chaining (CBC) con un vector de inicialización (IV) de 128 bits. El texto cifrado resultante tiene el prefijo IV. Si se incluye en la salida XML, está codificado en base64. Un ejemplo de método de cifrado AES es el siguiente:

```
<Método de cifrado
 Algoritmo="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
```

5.3 Algoritmos de cifrado de flujo

Los algoritmos de cifrado de flujo simple generan, en función de la clave, un flujo de bytes a los que se aplica XOR con los bytes de datos de texto sin formato para producir el texto cifrado en el cifrado y con los bytes de texto cifrado para producir texto sin formato al descifrar. Normalmente se utilizan para el cifrado de datos y se especifican por el valor del `Algorithm`atributo del `EncryptionMethod`hijo de un `EncryptedData` elemento.

NOTA: Es fundamental que cada clave de cifrado de flujo simple (o clave y vector de inicialización (IV) si también se usa un IV) se use solo una vez. Si alguna vez se usa la misma clave (o clave y IV) en dos mensajes, al aplicar XOR en los dos textos cifrados, puede obtener el XOR de los dos textos sin formato. Esto suele ser muy comprometedor.

En este documento no se especifican algoritmos de cifrado de flujo específicos, pero esta sección se incluye para proporcionar pautas generales.

Los algoritmos de transmisión suelen utilizar el `KeySize`parámetro explícito opcional. En los casos en los que el tamaño de la clave no sea evidente en el URI del algoritmo o en el origen de la clave, como en el uso de métodos de acuerdo de claves, este parámetro establece el tamaño de la clave. Si el tamaño de la clave que se utilizará es evidente y no está de acuerdo con el `KeySize`parámetro, DEBE devolverse un error. La implementación de cualquier algoritmo de flujo es opcional. El esquema para el parámetro KeySize es el siguiente:

```
Definición del esquema:

<nombre de tipo simple='Tipo de tamaño de clave'>
  <restricción base="entero"/>
</tipo simple>
```

5.4 Transporte clave

Los algoritmos de transporte de claves son algoritmos de cifrado de claves públicas especialmente especificados para cifrar y descifrar claves. Sus identificadores aparecen como `Algorithm`atributos de `EncryptionMethod`elementos hijos de `EncryptedKey`. `EncryptedKey`es a su vez hijo de un `ds:KeyInfo`elemento. El tipo de clave que se transporta, es decir el algoritmo en el que se prevé utilizar la clave transportada, viene dado por el `Algorithm`atributo del `EncryptionMethod`hijo `EncryptedData`o `EncryptedKey`padre de este `ds:KeyInfo`elemento.

(Los algoritmos de transporte de claves se pueden utilizar opcionalmente para cifrar datos, en cuyo caso aparecen directamente como el `Algorithm`atributo de un elemento `EncryptionMethod`secundario `EncryptedData`. Debido a que utilizan algoritmos de clave pública directamente, los algoritmos de transporte de claves no son eficientes para el transporte de ninguna cantidad de datos significativamente. más grandes que las claves simétricas.)

El algoritmo de transporte de claves RSA v1.5 que se proporciona a continuación son los que se utilizan junto con TRIPLEDES y la sintaxis de mensajes criptográficos (CMS) de S/MIME [algoritmos CMS]. El algoritmo de transporte de claves RSA v2 que se proporciona a continuación es el que se utiliza junto con AES y CMS [AES-WRAP].

5.4.1 RSA Versión 1.5

Identificador:
http://www.w3.org/2001/04/xmlenc#rsa-1_5 (REQUERIDO)

El algoritmo RSAES-PKCS1-v1_5, especificado en RFC 2437 [PKCS1], no toma parámetros explícitos. Un ejemplo de un `EncryptionMethod`elemento RSA versión 1.5 es:

```
<Método de cifrado
 Algoritmo="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
```

La `CipherValue`clave para dicha clave cifrada es la codificación base64 [MIME] de la cadena de octetos calculada según RFC 2437 [PKCS1 , sección 7.2.1: Operación de cifrado]. Como se especifica en la función EME-PKCS1-v1_5 RFC 2437 [PKCS1 , sección 9.1.2.1], el valor ingresado a la función de transporte de claves es el siguiente:

```
 CRIPTAR (PAD (TECLA))
```

donde el relleno tiene la siguiente forma especial:

```
02 | PD* | 00 | llave
```

where "|" is concatenation, "02" and "00" are fixed octets of the corresponding hexadecimal value, PS is a string of strong pseudo-random octets [RANDOM] at least eight octets long, containing no zero octets, and long enough that the value of the quantity being CRYPTed is one octet shorter than the RSA modulus, and "key" is the key being transported. The key is 192 bits for TRIPLEDES and 128, 192, or 256 bits for AES. Support of this key transport algorithm for transporting 192 bit keys is MANDATORY to implement. Support of this algorithm for transporting other keys is OPTIONAL. RSA-OAEP is RECOMMENDED for the transport of AES keys.

The resulting base64 [MIME] string is the value of the child text node of the `CipherData` element, e.g.


```
<CipherData> IWijxQjUrcXBYoCei4QxjWo9Kg8D3p9tlWoT4
   t0/gyTE96639In0FZFY2/rvP+/bMJ01EArmKZsR5VW3rwoPxw=
</CipherData>
```


5.4.2 RSA-OAEP

Identifier:
http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p (REQUIRED)

The RSAES-OAEP-ENCRYPT algorithm, as specified in RFC 2437 [PKCS1], takes three parameters. The two user specified parameters are a MANDATORY message digest function and an OPTIONAL encoding octet string `OAEPparams`. The message digest function is indicated by the `Algorithm` attribute of a child `ds:DigestMethod` element and the mask generation function, the third parameter, is always MGF1 with SHA1 (mgf1SHA1Identifier). Both the message digest and mask generation functions are used in the EME-OAEP-ENCODE operation as part of RSAES-OAEP-ENCRYPT. The encoding octet string is the base64 decoding of the content of an optional `OAEPparams` child element . If no `OAEPparams` child is provided, a null string is used.

```
Schema Definition:
     <!-- use these element types as children of EncryptionMethod
          when used with RSA-OAEP -->
     <element name='OAEPparams' minOccurs='0' type='base64Binary'/>
     <element ref='ds:DigestMethod' minOccurs='0'/>
```

An example of an RSA-OAEP element is:

```
<EncryptionMethod
   Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p">
  <OAEPparams> 9lWu3Q== </OAEPparams>
  <ds:DigestMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<EncryptionMethod>
```

The `CipherValue` for an RSA-OAEP encrypted key is the base64 [MIME] encoding of the octet string computed as per RFC 2437 [PKCS1, section 7.1.1: Encryption operation]. As described in the EME-OAEP-ENCODE function RFC 2437 [PKCS1, section 9.1.1.1], the value input to the key transport function is calculated using the message digest function and string specified in the `DigestMethod` and `OAEPparams` elements and using the mask generator function MGF1 (with SHA1) specified in RFC 2437. The desired output length for EME-OAEP-ENCODE is one byte shorter than the RSA modulus.

The transported key size is 192 bits for TRIPLEDES and 128, 192, or 256 bits for AES. Implementations MUST implement RSA-OAEP for the transport of 128 and 256 bit keys. They MAY implement RSA-OAEP for the transport of other keys.

5.5 Key Agreement

A Key Agreement algorithm provides for the derivation of a shared secret key based on a shared secret computed from certain types of compatible public keys from both the sender and the recipient. Information from the originator to determine the secret is indicated by an optional `OriginatorKeyInfo` parameter child of an `AgreementMethod` element while that associated with the recipient is indicated by an optional `RecipientKeyInfo`. A shared key is derived from this shared secret by a method determined by the Key Agreement algorithm.

Note: XML Encryption does not provide an on-line key agreement negotiation protocol. The `AgreementMethod` element can be used by the originator to identify the keys and computational procedure that were used to obtain a shared encryption key. The method used to obtain or select the keys or algorithm used for the agreement computation is beyond the scope of this specification.

The `AgreementMethod` element appears as the content of a `ds:KeyInfo` since, like other `ds:KeyInfo` children, it yields a key. This `ds:KeyInfo` is in turn a child of an `EncryptedData` or `EncryptedKey` element. The `Algorithm` attribute and `KeySize` child of the `EncryptionMethod` element under this `EncryptedData` or `EncryptedKey` element are implicit parameters to the key agreement computation. In cases where this `EncryptionMethod` algorithm URI is insufficient to determine the key length, a `KeySize` MUST have been included. In addition, the sender may place a `KA-Nonce` element under `AgreementMethod` to assure that different keying material is generated even for repeated agreements using the same sender and recipient public keys. For example:

```
 <EncryptedData>
   <EncryptionMethod Algorithm="Example:Block/Alg"
     <KeySize>80</KeySize>
   </EncryptionMethod>
   <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
     <AgreementMethod Algorithm="example:Agreement/Algorithm">
       <KA-Nonce>Zm9v</KA-Nonce>
       <ds:DigestMethod
       Algoritmo="http://www.w3.org/2001/04/xmlenc#sha1"/>
      <Información de clave del originador>
         <ds:ValorClave>....</ds:ValorClave>
       </OriginatorKeyInfo>
       <Información de clave del destinatario>
         <ds:ValorClave>....</ds:ValorClave>
       </RecipientKeyInfo>
     </Método de acuerdo>
   </ds:Información clave>
   <CipherData>...</CipherData>
</EncryptedData>
```

Si la clave acordada se utiliza para envolver una clave, en lugar de datos como se indica arriba, `AgreementMethod`aparecerá dentro de `ds:KeyInfo`un `EncryptedKey`elemento.

El esquema `AgreementMethod`es el siguiente:

```
Definición del esquema:

<elemento nombre="AgreementMethod" type="xenc:AgreementMethodType"/>
<complexType nombre="AgreementMethodType" mixto="verdadero">
  <secuencia>
    <elemento nombre="KA-Nonce" minOccurs="0" tipo="base64Binary"/>
    <!-- <elemento ref="ds:DigestMethod" minOccurs="0"/> -->
    <cualquier espacio de nombre="##other" minOccurs="0" maxOccurs="ilimitado"/>
    <elemento nombre="OriginatorKeyInfo" minOccurs="0"
             tipo="ds:KeyInfoType"/>
    <elemento nombre="RecipientKeyInfo" minOccurs="0"
             tipo="ds:KeyInfoType"/>
  </secuencia>
  <atributo nombre="Algoritmo" tipo="cualquierURI" uso="requerido"/>
</tipocomplejo>
```

5.5.1 Valores clave de Diffie-Hellman

Identificador:
http://www.w3.org/2001/04/xmlenc#DHKeyValue (OPCIONAL)

Las claves Diffie-Hellman pueden aparecer directamente dentro de `KeyValue` los elementos u obtenerse mediante `ds:RetrievalMethod`recuperaciones, además de aparecer en certificados y similares. El identificador anterior se puede utilizar como el valor del `Type`atributo de `Reference`o `ds:RetrievalMethod`elementos.

Como se especifica en [ESDH], una clave pública DH consta de hasta seis cantidades, dos números primos grandes p y q, un "generador" g, la clave pública y los parámetros de validación "seed" y "pgenCounter". Estos se relacionan de la siguiente manera: La clave pública = (g**x mod p) donde x es la clave privada correspondiente; p = j*q + 1 donde j >= 2. "seed" y "pgenCounter" son opcionales y se pueden utilizar para determinar si la clave Diffie-Hellman se ha generado de conformidad con el algoritmo especificado en [ESDH] . Debido a que los números primos y el generador se pueden compartir de forma segura entre muchas claves DH, es posible que se conozcan desde el entorno de la aplicación y son opcionales. El esquema para a `DHKeyValue`es el siguiente:

```
 Schema:

<element name="DHKeyValue" type="xenc:DHKeyValueType"/>
<complexType name="DHKeyValueType">
   <sequence>
      <sequence minOccurs="0">
         <element name="P" type="ds:CryptoBinary"/>
         <element name="Q" type="ds:CryptoBinary"/>
         <element name="Generator"type="ds:CryptoBinary"/>
      </sequence>
      <element name="Public" type="ds:CryptoBinary"/>
      <sequence minOccurs="0">
         <element name="seed" type="ds:CryptoBinary"/>
         <element name="pgenCounter" type="ds:CryptoBinary"/>
      </sequence>
   </sequence>
</complexType>
```

5.5.2 Acuerdo clave Diffie-Hellman

Identificador:
http://www.w3.org/2001/04/xmlenc#dh (OPCIONAL)

El protocolo de acuerdo de claves Diffie-Hellman (DH) [ESDH] implica la derivación de información secreta compartida basada en claves DH compatibles del remitente y el destinatario. Dos claves públicas DH son compatibles si tienen el mismo número principal y generador. Si, para el segundo, `Y = g**y mod p`entonces las dos partes pueden calcular el secreto compartido `ZZ = ( g**(x*y) mod p )`aunque cada una conozca sólo su propia clave privada y la clave pública de la otra parte. Los bytes cero iniciales DEBEN mantenerse para `ZZ`que tengan la misma longitud, en bytes, que `p`. El tamaño `p`DEBE ser de al menos 512 bits y `g`al menos 160 bits. Existen muchas otras consideraciones de seguridad complejas en la selección de `g`, `p`y aleatorio `x`como se describe en [ESDH].

El acuerdo clave Diffie-Hellman es opcional de implementar. Un ejemplo de un `AgreementMethod`elemento DH es el siguiente:

```
<Método de acuerdo
    Algoritmo="http://www.w3.org/2001/04/xmlenc#dh"
    ds:xmlns="http://www.w3.org/2000/09/xmldsig#">
  <KA-Nonce>Zm9v</KA-Nonce>
  <ds:Método de resumen
   Algoritmo="http://www.w3.org/2000/09/xmldsig#sha1"/>
  <Información de clave del originador>
    <ds:X509Data><ds:X509Certificado>
      ...
    </ds:X509Certificate></ds:X509Data>
  </OriginatorKeyInfo>
  <RecipientKeyInfo><ds:KeyValue>
    ...
  </ds:KeyValue></RecipientKeyInfo>
</Método de acuerdo>
```

Supongamos que el secreto compartido de Diffie-Hellman es la secuencia de octetos `ZZ`. El material de claves compartido necesario se calculará de la siguiente manera:

```
Material de codificación = KM(1) | KM(2) | ...
```

donde "|" es la concatenación de flujo de bytes y

```
KM(contador) = DigestAlg ( ZZ | contador | EncryptionAlg |
               KA-Nonce | Tamaño de clave)
```

`DigestAlg`
: El algoritmo de resumen de mensajes especificado por el `DigestMethod`hijo de `AgreementMethod`.

`EncryptionAlg`
: El URI del algoritmo de cifrado, incluidos los posibles algoritmos de ajuste de claves, en los que se utilizará el material de claves derivado ("Ejemplo: Bloque/Alg" en el ejemplo anterior), no el URI del algoritmo de acuerdo. Este es el valor del `Algorithm` atributo del `EncryptionMethod`hijo de `EncryptedData`o `EncryptedKey`abuelo de `AgreementMethod`.

`KA-Nonce`
: Base64 decodifica el contenido del `KA-Nonce`hijo de `AgreementMethod`, si está presente. Si el `KA-Nonce` elemento está ausente, es nulo.

`Counter`
: Un contador de un byte que comienza en uno y se incrementa en uno. Se expresa como dos dígitos hexadecimales donde las letras de la A a la F están en mayúsculas.

`KeySize`
: The size in bits of the key to be derived from the shared secret as the UTF-8 string for the corresponding decimal integer with only digits in the string and no leading zeros. For some algorithms the key size is inherent in the URI. For others, such as most stream ciphers, it must be explicitly provided.

For example, the initial `(KM(1))` calculation for the `EncryptionMethod` of the Key Agreement example (section 5.5) would be as follows, where the binary one byte counter value of 1 is represented by the two character UTF-8 sequence `01`, `ZZ` is the shared secret, and "`foo`" is the base64 decoding of "`Zm9v`".

```
SHA-1 ( ZZ01Example:Block/Algfoo80 )
```

Assuming that `ZZ` is `0xDEADBEEF`, that would be

```
SHA-1( 0xDEADBEEF30314578616D706C653A426C6F636B2F416C67666F6F3830 )
```

whose value is

```
0x534C9B8C4ABDCB50038B42015A181711068B08C1
```

Each application of `DigestAlg` for successive values of `Counter` will produce some additional number of bytes of keying material. From the concatenated string of one or more `KM`'s, enough leading bytes are taken to meet the need for an actual key and the remainder discarded. For example, if `DigestAlg` is SHA-1 which produces 20 octets of hash, then for 128 bit AES the first 16 bytes from `KM(1)` would be taken and the remaining 4 bytes discarded. For 256 bit AES, all of `KM(1)` suffixed with the first 12 bytes of KM(2) would be taken and the remaining 8 bytes of `KM(2)` discarded.

5.6 Symmetric Key Wrap

Symmetric Key Wrap algorithms are shared secret key encryption algorithms especially specified for encrypting and decrypting symmetric keys. Their identifiers appear as `Algorithm` attribute values to `EncryptionMethod` elements that are children of `EncryptedKey` which is in turn a child of `ds:KeyInfo` which is in turn a child of `EncryptedData` or another `EncryptedKey`. The type of the key being wrapped is indicated by the `Algorithm` attribute of `EncryptionMethod` child of the parent of the `ds:KeyInfo` grandparent of the `EncryptionMethod` specifying the symmetric key wrap algorithm.

5.6.1 CMS Key Checksum

Some key wrap algorithms make use of a key checksum as defined in CMS [CMS-Wrap]. The algorithm that provides an integrity check value for the key being wrapped is:

1. Compute the 20 octet SHA-1 hash on the key being wrapped.
2. Use the first 8 octets of this hash as the checksum value.

5.6.2 CMS Triple DES Key Wrap

Identifiers and Requirements:
: http://www.w3.org/2001/04/xmlenc#kw-tripledes (REQUIRED)

XML Encryption implementations MUST support TRIPLEDES wrapping of 168 bit keys and may optionally support TRIPLEDES wrapping of other keys.

An example of a TRIPLEDES Key Wrap `EncryptionMethod` element is as follows:

```
<EncryptionMethod
 Algorithm="http://www.w3.org/2001/04/xmlenc#kw-tripledes"/>
```

The following algorithm wraps (encrypts) a key (the wrapped key, WK) under a TRIPLEDES key-encryption-key (KEK) as adopted from [CMS-Algorithms]:

1. Represent the key being wrapped as an octet sequence. If it is a TRIPLEDES key, this is 24 octets (192 bits) with odd parity bit as the bottom bit of each octet.
2. Compute the CMS key checksum (section 5.6.1) call this CKS.
3. Let `WKCKS = WK || CKS`, where || is concatenation.
4. Generate 8 random octets [RANDOM] and call this IV.
5. Encrypt WKCKS in CBC mode using KEK as the key and IV as the initialization vector. Call the results TEMP1.
6. Let `TEMP2 = IV || TEMP1`.
7. Reverse the order of the octets in `TEMP2` and call the result `TEMP3`.
8. Encrypt `TEMP3` in CBC mode using the `KEK` and an initialization vector of `0x4adda22c79e82105`. The resulting cipher text is the desired result. It is 40 octets long if a 168 bit key is being wrapped.

The following algorithm unwraps (decrypts) a key as adopted from [CMS-Algorithms]:

1. Check if the length of the cipher text is reasonable given the key type. It must be 40 bytes for a 168 bit key and either 32, 40, or 48 bytes for a 128, 192, or 256 bit key. If the length is not supported or inconsistent with the algorithm for which the key is intended, return error.
2. Decrypt the cipher text with TRIPLEDES in CBC mode using the `KEK` and an initialization vector (IV) of `0x4adda22c79e82105`. Call the output `TEMP3`.
3. Reverse the order of the octets in TEMP3 and call the result `TEMP2`.
4. Decompose TEMP2 into IV, the first 8 octets, and `TEMP1`, the remaining octets.
5. Decrypt TEMP1 using TRIPLEDES in CBC mode using the `KEK` and the IV found in the previous step. Call the result `WKCKS`.
6. Decompose `WKCKS`. `CKS` is the last 8 octets and `WK`, the wrapped key, are those octets before the `CKS`.
7. Calculate a CMS key checksum (section 5.6.1) over the `WK` and compare with the `CKS` extracted in the above step. If they are not equal, return error.
8. `WK` is the wrapped key, now extracted for use in data decryption.

5.6.3 AES KeyWrap

Identifiers and Requirements:

http://www.w3.org/2001/04/xmlenc#kw-aes128 (REQUIRED)
http://www.w3.org/2001/04/xmlenc#kw-aes192 (OPTIONAL)
http://www.w3.org/2001/04/xmlenc#kw-aes256 (REQUIRED)

Implementation of AES key wrap is described below, as suggested by NIST. It provides for confidentiality and integrity. This algorithm is defined only for inputs which are a multiple of 64 bits. The information wrapped need not actually be a key. The algorithm is the same whatever the size of the AES key used in wrapping, called the key encrypting key or `KEK`. The implementation requirements are indicated below.

128 bit AES Key Encrypting Key
Implementation of wrapping 128 bit keys REQUIRED.
Wrapping of other key sizes OPTIONAL.
192 bit AES Key Encrypting Key
All support OPTIONAL.
256 bit AES Key Encrypting Key
Implementation of wrapping 256 bit keys REQUIRED.
Wrapping of other key sizes OPTIONAL.

Assume that the data to be wrapped consists of `N` 64-bit data blocks denoted `P(1)`, `P(2)`, `P(3)` ... `P(N)`. The result of wrapping will be `N+1` 64-bit blocks denoted `C(0)`, `C(1)`, `C(2)`, ... `C(N)`. The key encrypting key is represented by `K`. Assume integers `i`, `j`, and `t` and intermediate 64-bit register `A`, 128-bit register `B`, and array of 64-bit quantities `R(1)` through `R(N)`.

"|" represents concatentation so `x|y`, where `x` and `y` and 64-bit quantities, is the 128-bit quantity with `x` in the most significant bits and `y` in the least significant bits. `AES(K)enc(x)` is the operation of AES encrypting the 128-bit quantity `x` under the key `K`. `AES(K)dec(x)` is the corresponding decryption opteration. `XOR(x,y)` is the bitwise exclusive or of `x` and `y`. `MSB(x)` and `LSB(y)` are the most significant 64 bits and least significant 64 bits of x and y respectively.

If `N` is 1, a single AES operation is performed for wrap or unwrap. If `N>1`, then `6*N` AES operations are performed for wrap or unwrap.

The key wrap algorithm is as follows:

1. If `N` is `1`:
   - `B=AES(K)enc(0xA6A6A6A6A6A6A6A6|P(1))`
   - `C(0)=MSB(B)`
   - `C(1)=LSB(B)`

   If `N>1`, perform the following steps:
2. Initialize variables:
   - Set `A` to `0xA6A6A6A6A6A6A6A6`
   - For`i=1` to `N`,
     `R(i)=P(i)`
3. Calculate intermediate values:
   - For`j=0` to `5`,
     - For `i=1` to `N`,
       `t= i + j*N`
       `B=AES(K)enc(A|R(i))`
       `A=XOR(t,MSB(B))`
       `R(i)=LSB(B)`
4. Output the results:
   - Set `C(0)=A`
   - For `i=1` to `N`,
     `C(i)=R(i)`

The key unwrap algorithm is as follows:

1. If `N` is `1`:
   - `B=AES(K)dec(C(0)|C(1))`
   - `P(1)=LSB(B)`
   - If `MSB(B)` is `0xA6A6A6A6A6A6A6A6`, return success. Otherwise, return an integrity check failure error.

   If `N`>1, perform the following steps:
2. Initialize the variables:
   - `A=C(0)`
   - For `i=1` to `N`,
     `R(i)=C(i)`
3. Calculate intermediate values:
   - For `j=5` to `0`,
     - For `i=N` to `1`,
       `t= i + j*N`

```
B=AES(K)dec(XOR(t,A)|R(i))
A=MSB(B)
R(i)=LSB(B)
```

4. Output the results:
   - For `i=1` to `N`,
     `P(i)=R(i)`
   - If `A` is `0xA6A6A6A6A6A6A6A6`, return success. Otherwise, return an integrity check failure error.

For example, wrapping the data `0x00112233445566778899AABBCCDDEEFF` with the `KEK 0x000102030405060708090A0B0C0D0E0F` produces the ciphertext of `0x1FA68B0A8112B447`, `0xAEF34BD8FB5A7B82`, `0x9D3E862371D2CFE5`.

5.7 Message Digest

Message digest algorithms can be used in `AgreementMethod` as part of the key derivation, within RSA-OAEP encryption as a hash function, and in connection with the HMAC message authentication code method as described in [XML-DSIG].)

5.7.1 SHA1

Identifier:
http://www.w3.org/2000/09/xmldsig#sha1 (REQUIRED)

El algoritmo SHA-1 [SHA] no toma parámetros explícitos. Un ejemplo de un `DigestMethod`elemento SHA-1 es:

```
<Método de resumen
 Algoritmo="http://www.w3.org/2000/09/xmldsig#sha1"/>
```

Un resumen SHA-1 es una cadena de 160 bits. El contenido del `DigestValue`elemento será la codificación base64 de esta cadena de bits vista como un tren de octetos de 20 octetos. Por ejemplo, el `DigestValue`elemento para el resumen del mensaje:

```
A9993E36 4706816A BA3E2571 7850C26C 9CD0D89D
```

del Apéndice A del estándar SHA-1 sería:

```
<DigestValue>qZk+NkcGgWq6PiVxeFDCbJzQ2J0=</DigestValue>
```

5.7.2 SHA256

Identificador:
http://www.w3.org/2001/04/xmlenc#sha256 (RECOMENDADO)

El algoritmo SHA-256 [SHA] no toma parámetros explícitos. `DigestMethod`Un ejemplo de un elemento SHA-256 es:

```
<DigestMethod
 Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
```

Un resumen SHA-256 es una cadena de 256 bits. El contenido del `DigestValue`elemento será la codificación base64 de esta cadena de bits vista como un tren de octetos de 32 octetos.

5.7.3 SHA512

Identificador:
http://www.w3.org/2001/04/xmlenc#sha512 (OPCIONAL)

El algoritmo SHA-512 [SHA] no toma parámetros explícitos. `DigestMethod`Un ejemplo de un elemento SHA-512 es:

```
<Método de resumen
 Algoritmo="http://www.w3.org/2001/04/xmlenc#sha512"/>
```

Un resumen SHA-512 es una cadena de 512 bits. El contenido del `DigestValue`elemento será la codificación base64 de esta cadena de bits vista como un tren de octetos de 64 octetos.

5.7.4 RIPEMD-160

Identificador:
http://www.w3.org/2001/04/xmlenc#ripemd160 (OPCIONAL)

El algoritmo RIPEMD-160 [RIPEMD-160] no toma parámetros explícitos. Un ejemplo de un `DigestMethod` elemento RIPEMD-160 es:

```
<Método de resumen
 Algoritmo="http://www.w3.org/2001/04/xmlenc#ripemd160"/>
```

Un resumen RIPEMD-160 es una cadena de 160 bits. El contenido del `DigestValue`elemento será la codificación base64 de esta cadena de bits vista como un tren de octetos de 20 octetos.

5.8 Autenticación de mensajes

Identificador:
http://www.w3.org/2000/09/xmldsig# (RECOMENDADO)

La firma XML [XML-DSIG] es OPCIONAL para implementar en aplicaciones de cifrado XML. Es la forma recomendada de proporcionar autenticación basada en claves.

5.9 Canonicalización

Una canonicalización de XML es un método para serializar XML de forma coherente en un flujo de octetos, según sea necesario antes de cifrar XML.

5.9.1 Canonicalización inclusiva

Identificadores:
http://www.w3.org/TR/2001/REC-xml-c14n-20010315 (OPCIONAL)
http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments (OPCIONAL)

XML canónico [Canon] es un método de serialización de XML que incluye el espacio de nombres dentro del alcance y el contexto del atributo del espacio de nombres xml de los antepasados del XML que se está serializando.

Si XML se va a cifrar y luego descifrar en un entorno diferente y se desea preservar los enlaces de prefijo del espacio de nombres y el valor de los atributos en el espacio de nombres "xml" de su entorno original, entonces se debe utilizar la versión canónica XML con comentarios del XML. Sea la serialización que esté cifrada.

5.9.2 Canonicalización exclusiva

Identificadores:
http://www.w3.org/2001/10/xml-exc-c14n# (OPCIONAL)
http://www.w3.org/2001/10/xml-exc-c14n#WithComments (OPCIONAL)

Canonicalización XML exclusiva [Exclusivo] serializa XML de tal manera que incluye en la medida mínima práctica el enlace de prefijo de espacio de nombres y el contexto de atributo de espacio de nombres xml heredado de elementos ancestros.

Es el método recomendado donde se puede cambiar el contexto externo de un fragmento que fue firmado y luego cifrado. De lo contrario, la validación de la firma sobre el fragmento puede fallar porque la canonicalización mediante validación de firma puede incluir espacios de nombres innecesarios en el fragmento.

6 consideraciones de seguridad

6.1 Relación con las firmas digitales XML

La aplicación de cifrado y firmas digitales en partes de un documento XML puede dificultar el descifrado y la verificación de la firma posteriores. En particular, al verificar una firma se debe saber si la firma se calculó sobre la forma de elementos cifrados o no cifrados.

Un problema aparte, pero importante, es la introducción de vulnerabilidades criptográficas al combinar firmas digitales y cifrado sobre un elemento XML común. Hal Finney ha sugerido que cifrar datos firmados digitalmente, dejando la firma digital en claro, puede permitir ataques de adivinación de texto sin formato. Esta vulnerabilidad se puede mitigar mediante el uso de hashes seguros y nonces en el texto que se procesa.

De acuerdo con el documento de requisitos [EncReq], la interacción entre cifrado y firma es un problema de aplicación y está fuera del alcance de la especificación. Sin embargo, hacemos las siguientes recomendaciones:

1. Cuando los datos están cifrados, cualquier resumen o firma sobre esos datos debe cifrarse. Esto satisface el primer problema en el sentido de que sólo se pueden validar aquellas firmas que se pueden ver. También aborda la posibilidad de una vulnerabilidad de adivinación de texto sin formato, aunque puede que no sea posible identificar (o incluso conocer) todas las firmas de un determinado dato.
2. Emplee la transformación de firma "decrypt-except" [XML-DSIG-Decrypt] . Funciona de la siguiente manera: durante el procesamiento de transformación de firma, si encuentra una transformación de descifrado, descifre todo el contenido cifrado del documento excepto aquellos exceptuados por un conjunto enumerado de referencias.

Además, si bien las siguientes advertencias se refieren a inferencias incorrectas por parte del usuario sobre la autenticidad de la información cifrada, las aplicaciones deben disuadir la mala interpretación del usuario comunicando claramente qué información tiene integridad o está autenticada, es confidencial o no repudiable cuando se realizan múltiples procesos (por ejemplo, firma). y cifrado) y se utilizan algoritmos (por ejemplo, simétricos y asimétricos):

1. Cuando un sobre cifrado contiene una firma, la firma no necesariamente protege la autenticidad o integridad del texto cifrado [Davis].
2. Si bien la firma protege el texto sin formato, solo cubre lo que está firmado, los destinatarios de los mensajes cifrados no deben inferir la integridad o autenticidad de otra información sin firmar (por ejemplo, encabezados) dentro del sobre cifrado; consulte [XML-DSIG , 8.1.1 Solo lo que está firmado es seguro].

6.2 Información revelada

Cuando una clave simétrica se comparte entre varios destinatarios, esa clave simétrica *solo* debe usarse para los datos destinados a *todos* los destinatarios; Incluso si un destinatario no es dirigido a información destinada (exclusivamente) a otro en la misma clave simétrica, la información podría ser descubierta y descifrada.

Además, los diseñadores de aplicaciones deben tener cuidado de no revelar ninguna información en los parámetros o identificadores de algoritmos (por ejemplo, información en un URI) que debilite el cifrado.

6.3 Nonce y IV (Valor o Vector de Inicialización)

Una característica indeseable de muchos algoritmos de cifrado y/o sus modos es que el mismo texto sin formato, cuando se cifra con la misma clave, tiene el mismo texto cifrado resultante. Si bien esto no es sorprendente, invita a varios ataques que se mitigan al incluir datos arbitrarios y no repetidos (bajo una clave determinada) con el texto sin formato antes del cifrado. En los modos de encadenamiento de cifrado, estos datos son los primeros en cifrarse y, en consecuencia, se denominan IV (valor de inicialización o vector).

Los diferentes algoritmos y modos tienen requisitos adicionales sobre las características de esta información (por ejemplo, aleatoriedad y secreto) que afectan las características (por ejemplo, confidencialidad e integridad) y su resistencia a los ataques.

Dado que los datos XML son redundantes (por ejemplo, codificaciones Unicode y etiquetas repetidas) y que los atacantes pueden conocer la estructura de los datos (por ejemplo, DTD y esquemas), los algoritmos de cifrado deben implementarse y utilizarse cuidadosamente en este sentido.

Para el modo Cipher Block Chaining (CBC) utilizado por esta especificación, el IV no debe reutilizarse para ninguna clave y debe ser aleatorio, pero no es necesario que sea secreto. Además, en este modo, un adversario que modifique el IV puede realizar un cambio conocido en el texto sin formato después del descifrado. Este ataque se puede evitar asegurando la integridad de los datos de texto sin formato, por ejemplo firmándolos.

6.4 Denegación de Servicio

This specification permits recursive processing. For example, the following scenario is possible: `EncryptedKey` **A** requires `EncryptedKey` **B** to be decrypted, which itself requires `EncryptedKey` **A**! Or, an attacker might submit an `EncryptedData` for decryption that references network resources that are very large or continually redirected. Consequently, implementations should be able to restrict arbitrary recursion and the total amount of processing and networking resources a request can consume.

6.5 Unsafe Content

XML Encryption can be used to obscure, via encryption, content that applications (e.g., firewalls, virus detectors, etc.) consider unsafe (e.g., executable code, viruses, etc.). Consequently, such applications must consider encrypted content to be as unsafe as the unsafest content transported in its application context. Consequently, such applications may choose to (1) disallow such content, (2) require access to the decrypted form for inspection, or (3) ensure that arbitrary content can be safely processed by receiving applications.

7 Conformance

An implementation is conformant to this specification if it successfully generates syntax according to the schema definitions and satisfies all MUST/REQUIRED/SHALL requirements, including algorithm support and processing. Processing requirements are specified over the roles of ***decryptor***, ***encryptor***, and their calling ***application***.

8 XML Encryption Media Type

8.1 Introduction

XML Encryption Syntax and Processing [XML-Encryption] specifies a process for encrypting data and representing the result in XML. The data may be arbitrary data (including an XML document), an XML element, or XML element content. The result of encrypting data is an XML Encryption element which contains or references the cipher data.

The `application/xenc+xml` media type allows XML Encryption applications to identify encrypted documents. Additionally it allows applications cognizant of this media-type (even if they are not XML Encryption implementations) to note that the media type of the decrypted (original) object might be a type other than XML.

8.2 application/xenc+xml Registration

Este es un registro de tipo de medio tal como se define en Extensiones multipropósito de correo de Internet (MIME), parte cuatro: Procedimientos de registro [MIME-REG]

Nombre del tipo de medio MIME: aplicación

Nombre del subtipo MIME: xenc+xml

Parámetros requeridos: ninguno

Parámetros opcionales: juego de caracteres

Los valores permitidos y recomendados y la interpretación del parámetro charset son idénticos a los proporcionados para 'application/xml' en la sección 3.2 de RFC 3023 [XML-MT].

Consideraciones de codificación:

Las consideraciones de codificación son idénticas a las dadas para 'aplicación/xml' en la sección 3.2 de RFC 3023 [XML-MT].

Consideraciones de Seguridad:

Consulte la sección Consideraciones de seguridad [Cifrado XML] .

Consideraciones de interoperabilidad: ninguna

Especificación publicada: [Cifrado XML]

Aplicaciones que utilizan este tipo de medio:

XML Encryption es neutral en cuanto a dispositivos, plataformas y proveedores y es compatible con una variedad de aplicaciones web.

Información adicional:

Número(s) mágico(s): ninguno

Aunque no se puede contar con secuencias de bytes para identificar consistentemente los documentos XML Encryption, serán documentos XML en los que el elemento raíz ' `QName`s `LocalPart`es '`EncryptedData`'o ' `EncryptedKey`' con un nombre de espacio de nombres asociado de ' http://www.w3.org/2001/ 04/xmlenc# '. El `application/xenc+xml`nombre del tipo DEBE usarse solo para objetos de datos en los que el elemento raíz sea del

espacio de nombres de cifrado XML. Los documentos XML que contienen estos tipos de elementos en lugares distintos del elemento raíz se pueden describir utilizando funciones como [esquema XML].

Extensiones de archivo: .xml

Código(s) de tipo de archivo de Macintosh: "TEXTO"

Persona y dirección de correo electrónico a contactar para obtener más información:

José Reagle <reagle@w3.org>

Grupo de trabajo XENC <xml-encryption@w3.org>

Uso previsto: COMÚN

Autor/Cambiar controlador:

La especificación de cifrado XML es un producto del trabajo del World Wide Web Consortium (W3C), que tiene control de cambios sobre la especificación.

9 esquema y ejemplos válidos

Esquema
esquema-xenc.xsd
Ejemplo
enc-example.xml (no es criptográficamente válido pero ejercita gran parte del esquema)

10 referencias

TRIPLES
ANSI X9.52: Modos de operación del algoritmo de cifrado de datos triple. 1998.
AES
NIST FIPS 197: Estándar de cifrado avanzado (AES) . Noviembre de 2001.
http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
ENVOLTURA AES
RFC3394: Algoritmo de ajuste de claves del estándar de cifrado avanzado (AES) . J. Schaad y R. Housley. Informativo, septiembre de 2002.
Algoritmos CMS
RFC3370: Algoritmos de sintaxis de mensajes criptográficos (CMS) . R. Housley. Informativo, febrero de 2002.
http://www.ietf.org/rfc/rfc3370.txt
Envoltura CMS
RFC3217: Encapsulación de claves Triple-DES y RC2 . R. Housley. Informativo, diciembre de 2001.
http://www.ietf.org/rfc/rfc3217.txt
davis
Firma y cifrado defectuosos en S/MIME, PKCS#7, MOSS, PEM, PGP y XML. D. Davis. Conferencia Técnica Anual de USENIX. 2001.
http://www.usenix.org/publications/library/proceedings/usenix01/davis.html
DES
NIST FIPS 46-3: Estándar de cifrado de datos (DES). Octubre de 1999.
http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf
EncReq
Requisitos de cifrado XML . J. Reagle. Nota del W3C, marzo de 2002.
http://www.w3.org/TR/2002/NOTE-xml-encryption-req-20020304
ESDH
RFC 2631: Método de acuerdo de claves Diffie-Hellman. E. Rescorla. Seguimiento de estándares, 1999.
http://www.ietf.org/rfc/rfc2631.txt
http://www.w3.org/TR/2002/CR-xml-exc-c14n-20020212
Glosario
RFC 2828: Glosario de seguridad de Internet . R Shirey. Informativo, mayo de 2000.
http://www.ietf.org/rfc/rfc2828.txt
HMAC
RFC 2104: HMAC: hash con clave para autenticación de mensajes . H. Krawczyk, M. Bellare y R. Canetti. Informativo, febrero de 1997.
http://www.ietf.org/rfc/rfc2104.txt
HTTP
RFC 2616: Protocolo de transferencia de hipertexto - HTTP/1.1. J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach y T. Berners-Lee. Standards Track, junio de 1999.
http://www.ietf.org/rfc/rfc2616.txt
PALABRAS CLAVE
RFC 2119: Palabras clave para uso en RFC para indicar niveles de requisitos. S. Bradner. Mejores prácticas actuales, marzo de 1997.
http://www.ietf.org/rfc/rfc2119.txt
MD5
RFC 1321: Algoritmo de resumen de mensajes MD5. R. Rivest. Informativo, abril de 1992.
http://www.ietf.org/rfc/rfc1321.txt
MÍMICA
RFC 2045: Extensiones multipropósito de correo de Internet (MIME), primera parte: Formato de los cuerpos de los mensajes de Internet . N. Freed y N. Borenstein. Standards Track, noviembre de 1996.
http://www.ietf.org/rfc/rfc2045.txt
MIME-REG
RFC 2048: Extensiones multipropósito de correo de Internet (MIME), cuarta parte: Procedimientos de registro . N. Freed, J. Klensin y J. Postel. Mejores prácticas actuales, noviembre de 1996.
http://www.ietf.org/rfc/rfc2048.txt
NFC
TR15, formularios de normalización Unicode . M. Davis y M. Dürst. Revisión 18: noviembre de 1999.
http://www.unicode.org/unicode/reports/tr15/tr15-18.html .

Corrección NFC
"Corrigéndum n.º 2: Yod con normalización de Hiriq ".
http://www.unicode.org/versions/corrigendum2.html .
prop1
"Propuesta de hombre de paja de cifrado XML ". E. Simon y B. LaMacchia. Agosto de 2000.
http://lists.w3.org/Archives/Public/xml-encryption/2000Aug/0001.html
prop2
Otra propuesta de Cifrado XML . T. Imamura. Agosto de 2000.
http://lists.w3.org/Archives/Public/xml-encryption/2000Aug/0005.html
prop3
Sintaxis y procesamiento de cifrado XML . B. Dillaway, B. Fox, T. Imamura, B. LaMacchia, H. Maruyama, J. Schaad y E. Simon. Diciembre de 2000.
http://lists.w3.org/Archives/Public/xml-encryption/2000Dec/att-0024/01-XMLEncryption_v01.html
PKCS1
RFC 2437: PKCS #1: Especificaciones de criptografía RSA Versión 2.0. B. Kaliski y J. Staddon. Informativo, octubre de 1998.
http://www.ietf.org/rfc/rfc2437.txt
ALEATORIO
RFC 1750: Recomendaciones de aleatoriedad para la seguridad . D. Eastlake, S. Crocker y J. Schiller. Informativo, diciembre de 1994.
http://www.ietf.org/rfc/rfc1750.txt
RIPEMD-160
CryptoBytes, Volumen 3, Número 2. La función hash criptográfica RIPEMD-160 . Laboratorios RSA. Otoño de 1997.
ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto3n2.pdf
http://www.esat.kuleuven.ac.be/~cosicart/pdf/AB-9601/AB-9601.pdf
sha
Estándar de hash seguro . NIST FIPS 180-1. (RFC 3174). Abril de 1995.
http://www.itl.nist.gov/fipspubs/fip180-1.htm
Estándar de hash seguro. Borrador NIST FIPS 180-2 . 2001. (Ampliado para incluir SHA-384, SHA-256 y SHA-512)
http://csrc.nist.gov/encryption/shs/dfips-180-2.pdf
A Bin
R. Tobin. Conjunto de información para entidades externas , lista de correo XML Core, 2000 [solo miembro del W3C].
http://lists.w3.org/Archives/Member/w3c-xml-core-wg/2000OctDec/0054
UTF-16
RFC 2781: UTF-16, una codificación de ISO 10646. P. Hoffman y F. Yergeau. Informativo, febrero de 2000.
http://www.ietf.org/rfc/rfc2781.txt
UTF-8
RFC 2279: UTF-8, un formato de transformación de ISO 10646F. F. Yergeau. Standards Track, enero de 1998.
http://www.ietf.org/rfc/rfc2279.txt
URI
RFC 2396: Identificadores uniformes de recursos (URI): sintaxis genérica . T. Berners-Lee, R. Fielding y L. Masinter. Standards Track, agosto de 1998.
http://www.ietf.org/rfc/rfc2396.txt
http://www.ietf.org/rfc/rfc1738.txt
http://www.ietf.org/rfc/rfc2141.txt
RFC 2611: Mecanismos de definición de espacios de nombres URN. Mejores prácticas actuales. Daigle, D. van Gulik, R. Iannella, P. Falstrom. Junio de 1999.
http://www.ietf.org/rfc/rfc2611.txt
X509v3
Recomendación UIT-T X.509 versión 3 (1997). "Tecnología de la información - Interconexión de sistemas abiertos - El marco de autenticación de directorios" ISO/IEC 9594-8:1997.
XML
Lenguaje de marcado extensible (XML) 1.0 (segunda edición) . T. Bray, J. Paoli, CM Sperberg-McQueen y E. Maler. Recomendación del W3C, octubre de 2000.
Base XML
Base XML . J. Marsh. Recomendación del W3C, junio de 2001.
http://www.w3.org/TR/2001/REC-xmlbase-20010627/
XML-C14N
XML canónico. J. Boyer. Recomendación del W3C, marzo de 2001.
http://www.w3.org/TR/2001/REC-xml-c14n-20010315
http://www.ietf.org/rfc/rfc3076.txt
XML-exc-C14N
Canonicalización XML exclusiva . J. Boyer, D. Eastlake y J. Reagle. Recomendación del W3C, julio de 2002.
http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718/
XML-DSIG
Sintaxis y procesamiento de firmas XML . D. Eastlake, J. Reagle y D. Solo. Recomendación del W3C, febrero de 2002.
http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/
XML-DSIG-Descifrar
Transformación de descifrado para firma XML . M. Hughes, T. Imamura y H. Maruyama. Recomendación del W3C, diciembre de 2002.
http://www.w3.org/TR/2002/REC-xmlenc-decrypt-20021210
Cifrado XML
Sintaxis y procesamiento de cifrado XML . D. Eastlake y J. Reagle. Recomendación candidata del W3C, diciembre de 2002.
http://www.w3.org/TR/2002/CR-xmlenc-core-20020802/
Conjunto de información XML
Conjunto de información XML . J. Cowan y R. Tobin. Recomendación del W3C, octubre de 2001
http://www.w3.org/TR/2001/REC-xml-infoset-20011024/
XML-MT
RFC 3023: tipos de medios XML. M. Murata, S. St. Laurent y D. Kohn. Informativo, enero de 2001.
http://www.ietf.org/rfc/rfc2376.txt
XML-NS
Espacios de nombres en XML . T. Bray, D. Hollander y A. Layman. Recomendación del W3C, enero de 1999.
http://www.w3.org/TR/1999/REC-xml-names-19990114
esquema XML
Esquema XML Parte 1: Estructuras D. Beech, M. Maloney y N. Mendelsohn. Recomendación del W3C, mayo de 2001.

http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/
Esquema XML, parte 2: tipos de datos . P. Biron y A. Malhotra. Recomendación del W3C, mayo de 2001.
http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

XPath

Lenguaje de ruta XML (XPath) versión 1.0 . J. Clark y S. DeRose. Recomendación del W3C, octubre de 1999.
http://www.w3.org/TR/1999/REC-xpath-19991116